Desarrollo de aplicaciones peer-to-peer para iPhone

Mario Martínez Fidalgo

Titulo: Desarrollo de aplicaciones peer-to-peer para iPhone

Autor: Mario Martínez Fidalgo

Fecha: 26/06/2012

Director: Josep-Llorenç Cruz Díaz

Departamento del director: Departament d'Arquitectura de Computadors

Titulación: Ingeniería técnica en Informática de Gestión

Centro: Facultat d'Informàtica de Barcelona (FIB) Universidad: Universitat Politècnica de Catalunya (UPC) BarcelonaTech

ÍNDICE

1. INTRODUCCION 5
1.1. INFORME PREVIO 5
1.1.1. Introducción a las comunicaciones peer-to-peer 5
1.1.2. Los smartphones 5
1.1.3. Objetivos del proyecto 6
1.1.4. La elección 7
1.1.5. Caso de estudio 7
1.1.6. Objetivos personales 7
1.2. ESTRUCTURA DEL DOCUMENTO 8
2. ANALISIS 9
2.1. IPHONE 9
2.1.1. Historia 9
2.1.2. Hardware 10
2.1.3. Los otros 13
2.2. IOS 14
2.2.1. Interfaz 14
2.2.2. Teléfono 18
2.2.3. Aplicaciones 18
2.2.4. App Store 19
2.2.5. Actualizaciones 20
2.3. IOS SDK 20
2.3.1. Historia 21
2.3.2. Requisito 21
2.3.3. Contenido del SDK 22
2.3.4. Distribución 22
2.3.5. App Store 23
2.4. COCOS2D 23
2.5. OTRAS HERRAMIENTAS 24
2.5.1. Mac OS X 24
2.5.2. Gimp 25
2.5.3. Dropbox 25
3. DESARROLLO EN EL IOS 27
3.1. OBJECTIVE-C 27
3.1.1. Archivos 27
3.1.2. Clases 28
3.1.3. Métodos y mensajes 28
3.1.4. Declaración de propiedades 30
3.1.5. Protocolos 31
3.2. ARQUITECTURA DEL IOS 31
3.2.1. Cocoa Touch 33
3.2.2. Media Layer 33
3.2.3. Core Services Layer 33
3.2.4. Core OS Layer 33
3.3. ARQUITECTURA DE UNA APLICACIÓN 34
3.3.1. Patrón Modelo Vista Controlador 34
3.3.2. Objetos del núcleo de una aplicación 35
3.4. CICLO DE VIDA DE UNA APLICACIÓN 37
3.4.1. Inicio de una aplicación 38
3.4.2. Pasando a segundo plano 39
3.4.3. Volviendo al primer plano 39
3.4.4. Finalización de una aplicación 40
3.5. GESTIÓN DE MEMORIA 40
3.5.1. Introducción 40
3.5.2. Gestión de memoria 41
3.6. CONTROL TÁCTIL 43
4. COMUNICACIÓN 45
4.1. INTRODUCCIÓN 45
4.2. FRAMEWORK GAME KIT 46
4.2.1. Sesiones 46

4.2.2. Ejemplo de código de sesiones 50
4.2.3. Pruebas 51
4.3. LIMITACIONES DEL GAME KIT Y DEL BLUETOOTH 52
4.4. METODOLOGÍA DE DESARROLLO 53
4.4.1. Metodología de desarrollo en cascada 53
4.4.2. Decisión 54
4.5. ESPECIFICACIÓN 55
4.5.1. Definición del sistema 55
4.5.2. Análisis de requisitos 55
4.5.3. Especificación – casos de uso 58
4.5.4. Especificación – diagrama conceptual de datos 62
4.6. DISEÑO 63
4.6.1. Arquitectura de red 64
4.6.2. Protocolos 65
4.6.3. Protocolo de incorporación de un nuevo dispositivo a la red 66
4.6.4. Protocolo de envío – recepción de paquetes 68
4.6.5. Protocolo de desconexión de un dispositivo de la red 70
4.6.6. Protocolo de reconexión de un dispositivo 74
4.6.7. Diagrama de estados 76
4.6.8. Arquitectura en capas 77
4.6.9. Patrón de diseño – El patrón fachada 78
4.6.10. Diagrama de clases 79
4.7. CODIFICACIÓN - IMPLEMENTACIÓN 80
4.8. PRUEBAS 82
4.9. LIMITES Y FILOSOFÍA DE LA LIBRERÍA 83

5. JUEGO – PÓKER 85
5.1. INTRODUCCIÓN 85
5.2. METODOLOGÍA DE TRABAJO 85
5.3. ESPECIFICACIÓN 86
5.3.1. Definición del sistema 86
5.3.2. Análisis de requisitos 91
5.3.3. Especificación – casos de uso 94
5.3.4. Especificación – diagrama conceptual de los datos 98
5.4. DISEÑO 99
5.4.1. Utilización de la librería de P2P y su filosofía 100
5.4.2. Arquitectura en capas 100
5.4.3. Diagrama de clases 101
5.4.4. Patrón delegación 102
5.4.5. Patrón singleton 103
5.4.6. Seguridad 103
5.5. CODIFICACIÓN – IMPLEMENTACIÓN 104
5.6. PRUEBAS 104

6. COCOS2D – LA INTERFAZ DE USUARIO 107
6.1. INTRODUCCIÓN 107
6.2. COCOS2D 108
6.2.1. Introducción 108
6.2.2. CCDirector 108
6.2.3. CCNode 109
6.2.4. CCScene 110
6.2.5. CCLayer 110
6.2.6. CCSprite 110
6.2.7. CCLabel 111
6.2.8. CCMenu 111
6.3. DISEÑO 111
6.3.1. Distribución de las pantallas 111
6.3.2. Mapa navegacional de las pantallas 112
6.3.3. Criterios generales 113
6.4. PANTALLAS 115
6.4.1. Menú principal 115
6.4.2. Aviso del Bluetooth 116
6.4.3. Configuración de la partida 116
6.4.4. Buscar partida 116
6.4.5. Opciones 117
6.4.6. A cerca de 117
6.4.7. Lista de conectados 117

6.4.8. Lista de reconexión 118
6.4.9. Vista principal del juego 119
6.4.10. Cartas comunitarias 120
6.4.11. Turno 120
6.4.12. Apostar 121
6.4.13. Finalización de la mano 122
6.5. SONIDOS E IDIOMAS 122
6.6. PRUEBAS 123
7. PLANIFICACIÓN Y COSTES 125
7.1. INTRODUCCIÓN 125
7.2. PLANIFICACIÓN 125
7.2.1. Planificación inicial 126
7.2.2. Planificación final 127
7.3. COSTES 132
7.3.1. Recursos 132
7.3.2. Costes 133
7.3.3. Coste final 135
8. CONCLUSIONES 137
8.1. CONCLUSIÓN DEL PROYECTO 137
8.2. CONCLUSIÓN PERSONAL 137
8.3. MEJORAS Y AMPLIACIONES 138
9. AGRADECIMIENTOS 139
10. BIBLIOGRAFÍA 141

1. INTRODUCCION

1.1. Informe previo

1.1.1. Introducción a las comunicaciones peer-to-peer

La comunicación, proceso que tiene como objetivo la transmisión de información entre dos entidades[1], es una acción muy cotidiana realizada por los hombres, los animales y, a última instancia, las máquinas.

En el caso de las computadoras, la comunicación se realiza cuando están en red. Una red es un conjunto de equipos informáticos conectados entre sí por medio de dispositivos físicos que envían y reciben impulsos eléctricos, ondas electromagnéticas o cualquier otro medio para el transporte de datos para compartir información y recursos[2].

Un sistema distribuido se define como una colección de computadoras separadas físicamente y conectadas entre sí por una red; cada máquina posee sus componentes de hardware y software que el usuario percibe como un solo sistema (no necesita saber qué cosas están en qué máquinas). El usuario accede a los recursos remotos de la misma manera en que accede a recursos locales, o un grupo de computadores que usan un software para conseguir un objetivo en común[3]. En el paradigma de los sistemas distribuidos existes varias arquitecturas[4], entre ellas hay dos que son muy significativas para este proyecto, la de cliente-servidor y la de “peer-to-peer”.

La primera arquitectura define dos tipos de roles para las computadoras que están en red: el rol del cliente y el rol del servidor. Las computadoras que hacen el rol del cliente piden un recurso a la computadora que hace de servidor y esperan a recibir dicho recurso. En cambio, las computadoras que hacen el rol del servidor reciben las peticiones de los diferentes clientes y les envía el recurso que han pedido.

Por otra parte, la segunda arquitectura, “peer-to-peer”, se diferencia de la primera por realizar los dos roles en la misma computadora. Por tanto, cada computadora puede enviar y recibir peticiones de otras que estén conectadas en la misma red. Tal como indica una de las traducciones del termino inglés “peer”, todas las computadoras de la red se comportan igual.

1.1.2. Los smartphones

“Smartphone” es un término comercial para denominar a un teléfono móvil que ofrece más funciones que un teléfono celular común[5]. Las características comunes de los “smartphones” que se pueden encontrar actualmente en el mercado son las siguientes:

1 http://es.wikipedia.org/wiki/Comunicación
2 http://es.wikipedia.org/wiki/Red_de_computadoras
3 http://es.wikipedia.org/wiki/Computación_distribuida#Sistemas_distribuidos
4 http://en.wikipedia.org/wiki/Distributed_computing#Architectures
5 http://es.wikipedia.org/wiki/Smartphone

- Capacidad de arrancar un sistema operativo que permita ejecutar los mismos tipos de aplicaciones informáticas que las de una computadora.
- Capacidad de comunicación de voz y de datos por medio de la red telefónica tal como lo hace un teléfono móvil.
- Capacidad de conectividad para establecer una comunicación con otros dispositivos mediante conexiones inalámbricas como por el ejemplo el Wi-fi o el Bluetooth.
- Capacidad de interacción mediante el uso de interfaces de entrada, como por ejemplo teclados alfanuméricos o pantallas táctiles, para facilitar el acceso a la interfaz de usuario del sistema operativo. Actualmente la gran mayoría de "smartphones" incorporan pantallas táctiles.
- Son dispositivos de pequeñas dimensiones y ligeros, haciéndolos fáciles de transportar y de guardar en lugares pequeños, como por ejemplo en el bolsillo de un pantalón.
- Estos dispositivos de pequeñas dimensiones suelen ir determinados por el tamaño de la pantalla debido a que este ocupa la mayor parte de la superficie (en los casos de no tener teclado alfanumérico). Se pueden encontrar dispositivos con pantallas desde 2.8 pulgadas hasta 5 pulgadas. Los dispositivos con pantallas más grandes de 5 pulgadas y con características similares a las descritas anteriormente se denominan "tablets".

Actualmente los "smartphones" son muy populares, a diferencia de hace unos cuantos años, y su integración en la sociedad va aumentado a pasos agigantados. Pero no todo son ventajas y debido a sus características, pueden acarrean también una serie de desventajas o limitaciones. El principal causante de la mayoría de las limitaciones es el tamaño reducido de este. En la siguiente lista se muestra cuales son estas limitaciones:

- Menores prestaciones respecto a una computadora de sobremesa. En otras palabras, llevan procesadores menos potentes, menos memoria, menor capacidad gráfica, etc. Por lo tanto, en algunas aplicaciones podremos ver algunas diferencias entre la versión "smartphone" y la versión de computadora. Naturalmente, en aplicaciones como por ejemplo un cliente de correo o un visor de documentos no veremos prácticamente ninguna diferencia, pero en el caso de los videojuegos la diferencia es muy evidente.
- Poca autonomía, menor que un teléfono celular común. En muchos casos, la autonomía de estos dispositivos suele ser de poco más de un día. Naturalmente, esta cifra puede variar, pero con un uso medio como tener la tarifa de datos en funcionamiento, realizar varias llamadas, utilizar un rato el Wi-fi o Bluetooth, escribir emails, navegar por internet y utilizar otras aplicaciones, hacen que su duración sea poco más de un día.

En la actualidad existe una gran variedad de "smartphones" tanto por marcas —Apple, HTC, Google, Microsoft, RIM, Palm, Nokia, Motorola, Samsung, Sony Ericsson,...— como por sistemas operativos —iOS(antiguamente iPhone OS), Android, Windows Phone 7, BlackBerry OS, webOS, Symbian,...— .

1.1.3. Objetivos del proyecto

Una vez hecha la introducción de los conceptos "peer-to-peer" y "smartphone", conceptos

fundamentales que definen este proyecto final de carrera, se pasa a exponer los objetivos del proyecto:

- Diseñar y desarrollar una librería estática que trabaje sobre el framework Game Kit —librería que contiene el código que realiza la conexión con otro iPhone— para que permita crear una red de iPhones. Esta red debe conseguir el máximo de dispositivos conectados posibles y que las conexiones sean recuperables en caso de desconexiones. Además esta librería debe ser reutilizable para otras aplicaciones.
- Aplicar la librería desarrollada en el desarrollo de una aplicación real para verificar la viabilidad de ésta.

1.1.4. La elección

Ahora que ya se ha presentado los objetivos del proyecto, solo falta explicar porque se ha escogido el iPhone como el "smartphone" en el que se enfocará el siguiente proyecto final de carrera.

Entre todos los sistemas operativos vistos en el final del apartado "smartphones", los más famosos son Android y iOS, y la elección estaba entre estos dos.

La elección fue a favor de iOS principalmente por interés personal, ya que anteriormente había desarrollado en plataformas de Microsoft o de Java, pero nunca en una de Apple. Además, los dispositivos con iOS tienen una mayor aceptación social en España y, en menor medida, en Europa que el conjunto de dispositivos con Android. Este argumento coge aún más fuerza si se considera que un factor de éxito, para una aplicación que se basa en la comunicación con muchos dispositivos, es la mayor posibilidad de encontrarse con personas que tengan el mismo sistema operativo (iOS).

1.1.5. Caso de estudio

En este proyecto, para poder evaluar las posibilidades de comunicación entre iPhones, es necesario el desarrollo de una aplicación en concreto. Entre todos los tipos de aplicaciones que existen, uno de los más adecuado para desarrollar es un juego, porque puede permitir jugar más de un jugador en una misma partida. Además, los juegos que permiten jugar varios jugadores reales, y no programados, son más dinámicos y divertidos que los juegos de un solo jugador.

Entre los diferentes tipos de juegos que existen, se ha decidido escoger uno que fuera conocido y actual, el Texas Hold'em. Este juego es una variante del póker clásico que tiene la particularidad de jugar con cartas comunitarias y descubiertas. El póker es un juego de cartas y apuestas que tiene el objetivo de conseguir la mayor cantidad posible de puntos o de dinero de los demás jugadores. Para este juego, es necesario la comunicación entre los jugadores-dispositivos para obtener las cartas que le toca a cada jugador y para poder realizar las apuestas.

1.1.6. Objetivos personales

Para cumplir los objetivos que se han establecido, es necesario cumplir previamente una serie

de condiciones personales para poder desarrollarlos. Estas condiciones están descritas en la siguiente lista:

- Aprender el lenguaje de programación Objective-C y los frameworks necesarios del iOS SDK[6].
- Investigar y experimentar con el framework Game Kit del iOS SDK.
- Aprender a usar el framework de código libre Cocos2d para iPhone. Este framework está especializado en la creación de gráficos, enfocado básicamente en la creación de videojuegos, aunque también se puede utilizar para la creación de cualquier otro tipo de aplicación.

1.2. Estructura del documento

Para poder tener una visión global del contenido de este documento, en este apartado se va resumir los diferentes temas que se va tratar en cada uno de los capítulos.

En el capítulo 2, Análisis, se tratará de las herramientas necesarias para desarrollar el proyecto. Las herramientas que serán analizadas van a ser el iPhone, su sistema operativo -el iOS- , el SDK del iOS, el framework Cocos2d entre otras herramientas.

A continuación, el capítulo 3, Desarrollo en el iOS, se centrará en los principales conceptos de desarrollo de aplicaciones para el iOS, como por ejemplo las diferentes arquitecturas que intervienen, el lenguaje de programación, el ciclo de vida de una aplicación, etc.

Los capítulos centrales de este documento son el 4 y el 5, Comunicación y Juego – Poker. Cada uno de estos capítulos se centrará en exponer todas las etapas de desarrollo partiendo de una determinada metodología de desarrollo. Estos dos capítulos corresponden a los dos objetivos del proyecto: el desarrollo de la librería de comunicación que permita crear una red de iPhones y el desarrollo de una aplicación para verificar el funcionamiento de la librería de comunicación.

El capítulo 6, Cocos2d – la interfaz de usuario, tratará de una introducción al framework Cocos2d y, por otra parte, el desarrollo de la interfaz de usuario utilizando este framework.

A continuación, en el capítulo 7, Planificación y costes, tal como indica su nombre, tratará de exponer como ha sido planificado la ejecución de las diferentes tareas del proyecto y cuales han sido los costes que se han derivado del desarrollo del proyecto.

Y por último, en el capítulo 8, Conclusión, se valorará si se han conseguido los objetivos establecidos y las diferentes conclusiones que se ha podido obtener de la realización de este proyecto.

[6] Software Development Kit o, traducido al castellano, kit de desarrollo de software.

2. ANALISIS

En el capítulo anterior se ha visto, entre otras cosas, de que trata este proyecto final de carrera, partiendo de una explicación básica de conceptos y acabando en la definición de los objetivos que se quiere llegar. Para alcanzar estos objetivos será necesario el uso de una serie de herramientas.

En este capítulo se realizará un análisis detallado de las diferentes herramientas utilizadas para llevar a cabo este proyecto. El capítulo estará dividido en los siguientes apartados: iPhone, iOS, iOS SDK, Cocos2d y otras herramientas.

2.1. iPhone

El iPhone es una familia de "smartphones" con capacidad multimedia, pantalla multitáctil capacitiva (sin puntero) y un sistema operativo que utiliza un interfaz con un diseño minimalista (tendencia artística surgida en Estados Unidos que reduce las obras a sus formas o estructuras geométricas, buscando la máxima expresión con los mínimos medios[7]) diseñado por la empresa Apple.

2.1.1. Historia

En el 9 de enero del 2007, durante el Macworld 2007 y tras varios meses de rumores, Steve Jobs (CEO de Apple) anunció el iPhone. Esta primera versión fue inicialmente lanzada solo para Estados Unidos el 29 de junio, aunque posteriormente se extendió a los países europeos de Reino Unido, Francia y Alemania el 9 de noviembre del mismo año.

Imagen 2.1 - Steve Jobs durante la presentación del iPhone en el Macworld 2007

[7] http://www.wordreference.com/definicion/minimalismo

Al año siguiente, exactamente el 9 de junio de 2008 y durante el WWDC 2008 (Worldwide Developers Conference), fue presentado iPhone 3G. Al mes siguiente, el 11 julio, se inició la comercialización de la segunda versión del iPhone en 22 países, una gran ampliación respecto a los 6 países de la primera versión, y que acabó en 70 países a finales de año. Junto con el lanzamiento del iPhone 3G, se lanzó el servicio llamado App Store para iPhone. Este nuevo servicio permitía a los usuarios buscar y descargar nuevas aplicaciones. Gracias a la llegada internacional de iPhone 3G y la aparición del App Store, la plataforma de Apple consiguió un impulso muy importante para su éxito definitivo.

En los siguientes años, Apple continuó con los mismos movimientos realizados en el año 2008 para introducir las nuevas versiones del iPhone (presentación en el WWDC, en junio, y lanzamiento comercial al mes siguiente, en julio). De esta manera, se presentó y se comercializó en el año 2009 el iPhone 3GS y, en el año 2010, el iPhone 4. No obstante, esta costumbre fue rota en el año 2011 con la presentación de la última versión, el iPhone 4S. Esta última versión fue presentado el 4 de octubre de 2011 (cuatro meses más tarde de lo habitual) en el evento “Let's talk iPhone” (Hablemos del iPhone).

2.1.2. Hardware

Pantalla e interfaz

La pantalla es el principal interfaz del iPhone, desde esta pantalla multitáctil capacitiva se da todas las ordenes usando gestos complejos con los dedos. Al ser la pantalla de tipo capacitiva se evita el uso de un “styluses” (sólo un punto de contacto a la vez) y permitiendo solo su uso mediante los dedos (varios puntos de contacto a la vez, es decir multitáctil) . Mediante los gestos complejos el usuario será capaz de mover el contenido en cualquier dirección y desplazarse a través de los diferentes menús. Además, gracias a que la interfaz simula la física de un objeto real, el usuario consigue interacción más natural con la ayuda de los efectos visuales que se producen mientras se interactúa con los menús.

La pantalla reacciona a la información que recibe tres sensores. El primero es un sensor de proximidad que hace la función de apagar y bloquear la pantalla cuando se pone el iPhone cerca de la cara, de esta manera se conseguir ahorrar batería y prevenir que la piel de la cara o de la oreja interactúe con los menús. El segundo es un sensor de la luz ambiental que permite ajustar automáticamente el brillo de la pantalla según la cantidad de luz, haciendo que la vista del usuario esté protegida y que el iPhone ahorre batería. El tercer sensor es un acelerómetro de tres ejes que permite detectar la orientación del iPhone y que permite, por ejemplo, reorientar el contenido de la pantalla según se tenga orientada la pantalla en forma vertical o en forma horizontal.

Los únicos botones que dispone el iPhone son cuatro. El botón principal del iPhone es el de inicio que está situado debajo de la pantalla y que realiza las funciones de ir al menú principal (Home), salir de una aplicación, ir al buscador spotlight (con un click) o entrar al menú de multitarea (con dos clicks). El siguiente botón es el de encender/apagar/bloquear el dispositivo y está situado en el

costado superior. Y en el lateral izquierdo encontramos los dos últimos botones, el de silencio y el de subir/bajar el volumen.

Imagen 2.2 - iPhone 4

Sonido y salidas

En la parte inferior del iPhone se encuentra un altavoz y un micrófono, situados alrededor del conector de la base. Por encima de la pantalla se sitúa el auricular del teléfono. El iPhone 4 incluye un nuevo micrófono que tiene como objetivo cancelar el ruido de fondo que se puede producir durante una llamada. El iPhone, como dispositivo multimedia, también incluye un conector de 3,5 mm para colocar auriculares que está situado en la parte superior izquierda. Tal como se describió en el apartado anterior, en el costado izquierdo incluye un control de volumen. Además del altavoz y del conector de 3,5 mm, se puede utilizar el Bluetooth para escuchar audio inalámbricamente en auriculares y altavoces. Hay que remarcar que esta es la única función del Bluetooth ya que, a diferencia de la gran mayoría de móviles, este no puede transmitir archivos multimedia (música, imágenes, videos, …) por carecer del protocolo de transferencia de archivos OBEX.

Batería

El iPhone dispone de una batería interna recargable que no es sustituible por el usuario, a diferencia de la mayoría de teléfonos móviles. Esta característica es bastante común en los productos de Apple y ha sido muy criticada desde hace muchos años. Aun así, Apple tiene un servicio de substitución de baterías y un auto kit de substitución con los materiales necesarios e instrucciones. Por otra parte, a través del conector inferior del iPhone se puede recargar el dispositivo mediante un cable USB. La recarga se puede realizar mientras el cable está conectado a una computador para sincronizar su contenido o mediante el uso de un adaptador-cargador de pared para cargar en una toma de CA.

Cámara

Las dos primeras versiones del iPhone (el original y el 3G) disponían una cámara de 2.0 megapíxeles en la parte posterior. Esta primera versión no tenía enfoque automático, ni flash y ni grabación de video. En la siguiente versión, el 3Gs, aumentó las prestaciones de la cámara con la introducción de más megapíxeles (3.2), enfoque automático, grabación de video (resolución VGA). En el iPhone 4 se volvió aumentar las prestaciones con 5 megapíxeles en la cámara, la introducción de un flash de tipo LED, grabación en resolución de 720p y la introducción de una segunda cámara con resolución VGA en la parte delantera para poder realizar video-llamadas. Continuando la misma línea, la última versión del iPhone, el iPhone 4S, fue presentado con más megapíxeles (8 megapíxeles) y más resolución (1080p)

Almacenamiento y SIM

Todas las versiones del iPhone incluyen una unidad de almacenamiento interno de tipo flash. El tamaño de esta unidad ha ido aumentado versión tras versión, comenzando con la primera versión que tenía dos variantes de 4 GB y 8 GB, acabando con el iPhone 4S que tiene las variantes de 16 GB, 32 GB y 64 GB. A diferencia de la mayoría de "smartphones" del mercado, el iPhone no tiene ranura para utilizar tarjetas de memoria.

Además de existir variantes de almacenamiento en una misma versión del iPhone, también existen variantes por el tipo de red telefónico. Las variantes de red son el GSM y la CDMA. En el GSM, el iPhone incorpora una bandeja para introducir la tarjeta SIM que está situada en la parte superior del dispositivo en las tres primeras versiones y situada en la parte derecha en el iPhone 4. En el iPhone 4, además del cambio de ubicación de la bandeja, cambió la versión de la tarjeta SIM, utilizando una versión más reducida llamada MicroSim. En cambio, la variante con CDMA no utilizan tarjeta SIM, como en todos los teléfonos que utiliza esta red, y por lo tanto no tiene ninguna bandeja.

Conectividad

La conectividad del iPhone se divide en dos grupos de componentes, los que permiten conectarse a otros dispositivos y los que permiten conectarse a internet.

En el primer grupo consigue conectarse a otros dispositivos mediante el Bluetooth, pero tal como se comentó en el subapartado de Sonidos y salidas, este solo se conecta a cascos y altavoces inalámbricos. Pero además de estos dos tipos de dispositivos, hay una tercera opción aun no comentada, la conexión a otros iPhones. Esto es posible gracias a la liberación de un framework del SDK[8] del iPhone que permite la transmisión de datos entre diferentes iPhones.

El otro grupo de conectividad está formado por el Wi-Fi y la banda de datos de la red telefónica. La diferencia entre estos dos consiste en que en la banda de datos de la red obtiene

[8] Software Development Kit o, traducido al castellano, kit de desarrollo de software.

acceso a internet directamente, mientras que en el Wi-Fi necesita que la red local Wi-Fi que se haya conectado tenga acceso a internet (que no siempre es así).

Tabla de las diferentes versiones del iPhone

Imagen 2.3 – Tabla de las diferentes versiones del iPhone

2.1.3. Los otros

Basándose en el concepto inicial del iPhone, Apple lanzó nuevos productos con características muy similares al iPhone. Estos productos son el iPod Touch y el iPad.

El iPod Touch básicamente es igual al iPhone, tanto características del hardware (componentes internos y dimensiones del dispositivo) como el software, con la única diferencia en que este no tiene funciones de teléfono (tampoco tiene acceso a la banda de datos para acceder a internet). Al no tener funciones de teléfono, el iPod Touch se considera un reproductor multimedia. Existe exactamente las mismas versiones que el iPhone, saliendo tres meses más tarde que la nueva versión del iPhone.

Imagen 2.4 – iPod Touch cuarta generación

El iPad sigue la misma idea que el iPod Touch aunque con enfoque diferente, en este caso enfocado al mundo de las “tablets”. Al ser una “tablet”, las dimensiones de la pantalla son distintas, ya que se pasa de las 3.5”a las 9.7”. Aun así, conserva la misma filosofía que su hermano menor manteniendo los mismo componentes internos del hardware y el mismo software de la misma generación del iPhone. Actualmente (2011) se comercializa la segunda generación, el iPad 2.

Imagen 2.5 – iPad 2

2.2. iOS

El iOS es el sistema operativo que se ejecuta en el iPhone, el iPod Touch y el iPad. Antiguamente el nombre que tenía el sistema operativo era iPhone OS, pero con la diversificación de productos que lo utilizaban decidieron cambiarlo a un nombre más general. El núcleo del sistema operativo es una variante del núcleo Darwin, núcleo que utilizado en Mac OS X (el sistema operativo que utiliza las computadoras de Apple). El iOS soporta la instalación de aplicaciones desarrolladas por Apple al igual que desarrolladas por terceros. De la misma manera que pasaba con la familia iPod, la única manera de gestionar los contenidos del iOS desde una computadora es utilizando la aplicación iTunes (actualmente disponible para la plataforma Mac y Windows). El iTunes no sólo permite la gestión de música y vídeos, sino que, entre otras cosas, también permite realizar las actualizaciones del iOS que son proporcionadas gratuitamente por Apple.

2.2.1. Interfaz

La interfaz sigue la misma filosofía de todos los productos de Apple, el uso del diseño minimalista, reflejado principalmente en el diseño de la pantalla de inicio. En esta pantalla se caracteriza principalmente por mostrar simplemente una lista gráfica de todas las aplicaciones disponibles. Aun así, se puede apreciar tres zonas diferentes en la pantalla de inicio. La primera es una barra informativa en la parte superior que muestra la señal de la red telefónica (en el caso de los iPhones), la disponibilidad de red de datos (a través de la banda de datos de la red telefónica o del

Wi-Fi), la hora y el estado de la batería. La segunda zona y principal muestra todas las aplicaciones disponibles en el dispositivo, como máximo puede mostrar 16 aplicaciones y, en caso de haber más de 16, el acceso a las demás se realiza mediante el gesto de pasar página, mostrando una nueva página y así hasta un máximo de 11 páginas más. Y la tercera zona es una bandeja o base de aplicaciones fijas (máximo 4) respecto al paso de las páginas de aplicaciones de la segunda zona, dando siempre acceso directo a estas aplicaciones. Las aplicaciones pueden moverse entre las diferentes zonas (principal y la bandeja) y páginas, además de ser quitadas.

Otra característica fundamental de la interfaz es que casi todas las entradas se realizan con gestos complejos reconocidos a través de la pantalla multitáctil. Gracias a esta técnica, el usuario puede mover el contenido en cualquier dirección con un simple toque y arrastres de los dedos. Un ejemplo más concreto de un gesto complejo es el gesto de pellizcar permitiendo que en aplicaciones como el navegador web o el visualizador de fotos puedan realizar el zoom.

Como regla general y no obligatoria, Apple propone que la interfaz de las aplicaciones siga una filosofía común para la navegación de contenidos, pantallas y menús. La razón de esta propuesta es quitar al usuario la necesidad de aprender el manejo de una nueva interfaz al encontrarse ante una nueva aplicación.

Imagen 2.6 – barra de navegación

En la imagen 2.6 se puede ver un ejemplo de navegación entre menús a través de listados de contenidos y una barra de navegación. En la primera pantalla se puede observar en la barra de navegación el nombre de la pantalla “Groups” y, debajo, el listado vertical con todos los diferentes grupos que hay. En la parte derecha de cada grupo se observa una flecha, la cual indica que seleccionando ese grupo se navegará al menú del correspondiente grupo, tal como se puede observar en la segunda imagen. Los cambios que hay en la segunda pantalla son la aparición de los botones de navegación para volver al menú anterior (en la parte superior izquierda) y que lleva al menú de añadir un nuevo contenido al grupo (en la parte superior derecha), y la aparición de otro tipo de listado. Este nuevo listado se caracteriza por agrupar el contenido en grupos de apellidos que

comienzan con la misma letra, por tener un buscador al inicio del listado y por tener una barra de acceso directo a un determinado grupo en la parte derecha del listado. Al igual que pasaba en la primera pantalla, al seleccionar un contenido, este te lleva a su correspondiente menú. En la tercera pantalla se puede observar que la barra de navegación no ha cambiado mucho, conserva el botón de navegación hacia el menú anterior (indicando el nombre del menú hacia donde se navega) y mantiene el botón en la parte derecha, pero esta vez está destinado a activar el modo edición del menú. En cambio el listado de contenidos es diferente, mantiene la idea de hacer grupos, pero esta vez el aspecto es totalmente diferente al no ocupar todo el espacio de la pantalla y dejando mostrar un fondo. La desaparición del buscador y la barra de índices de grupos viene motivado a que el contenido de esta pantalla muestra un listado de información fija y reducida que ha hecho innecesaria su utilización.

Otro punto importante de esta filosofía es que el número de pantallas o menús por donde se pueda navegar no sea muchas, porque si no se corre el peligro de que el usuario no sepa en qué punto de la aplicación se encuentra.

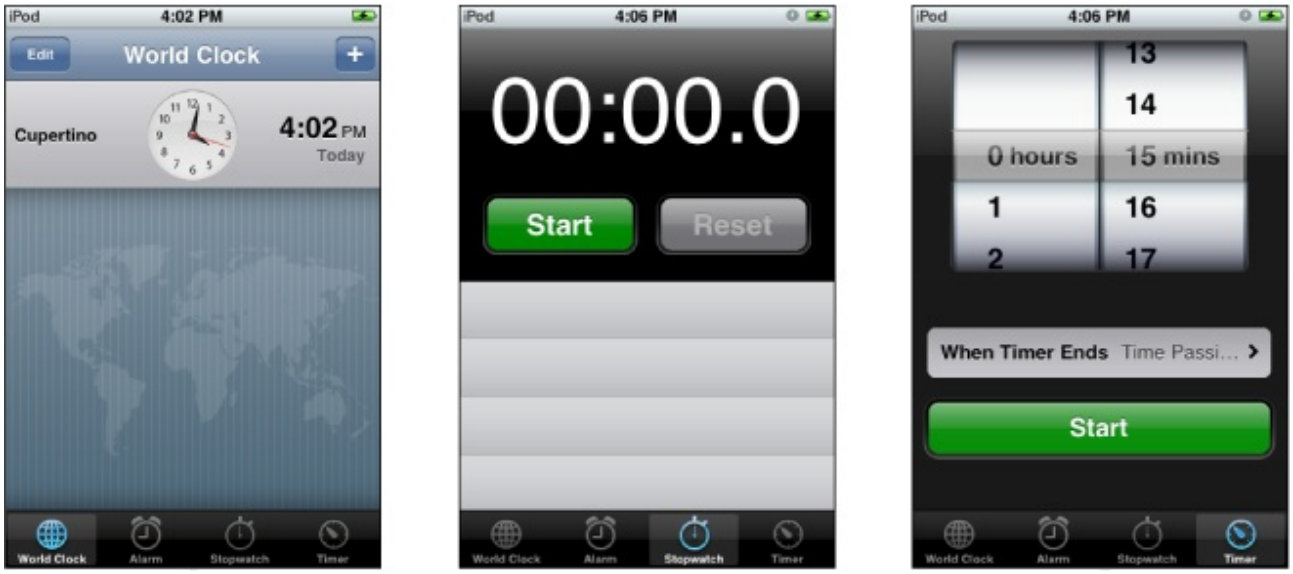

Imagen 2.7 – barra de pestañas

Otro ejemplo de estilo de navegación es la que se ve en la imagen 2.7, basada en la navegación mediante una barra con pestañas que se encuentra en la parte inferior de la pantalla. A diferencia de la barra de navegación vista en la imagen 2.6, esta navegación es más directa ya que no hace falta pasar por un menú antes de ir a otro, y no hace falta que la navegación sea sobre el mismo contexto, aquí se puede pasar, por ejemplo, del menú de la hora de cada país a otro menú tan diferente como es un cronometro.

La visualización de la mayoría de aplicaciones oficiales de Apple es en posición vertical. De esta manera se puede desplazar a través de los menús, como los vistos en las imágenes 2.6 y 2.7, deslizando con un solo dedo sobre la pantalla de forma que teniendo el iPhone o el iPod Touch en posición vertical podamos trabajar con él con una sola mano (utilizando los toques del dedo pulgar), exactamente de la misma forma que haríamos con un móvil tradicional. Hay dos razones por las que

es mejor tenerlo en posición vertical, la primera es que hay mejor agarre de la mano al dispositivo en posición vertical que horizontal y la segunda razón es que utilizando el dispositivo con una sola mano el pulgar de esa mano no puede llegar a todas las partes de la pantalla si está en posición horizontal y, en cambio, sí puede cuando está en vertical.

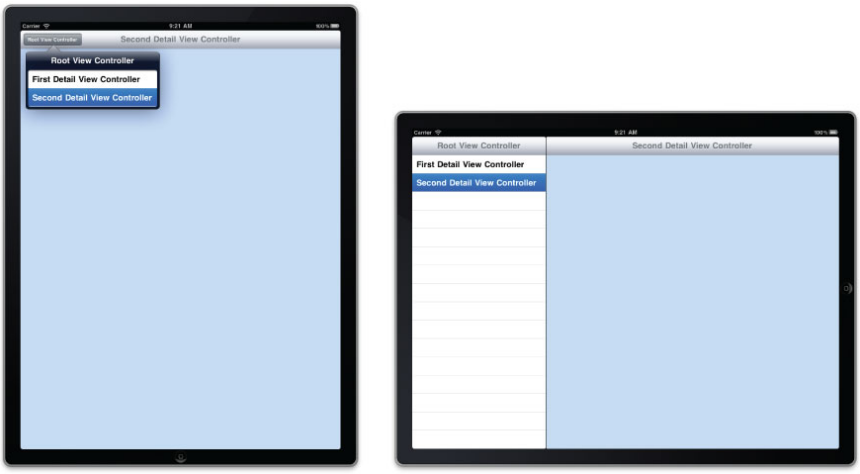

Imagen 2.7 – interfaz del iPad

Esta filosofía cambia un poco en el caso del iPad, ya que han modificado el interfaz para aprovechar el mayor tamaño de la pantalla. Tal como se puede ver en la imagen 2.7, uno de esos cambios consiste en dividir la pantalla en dos cuando está en posición horizontal, teniendo en un lado un listado de enlaces a los diferentes menús y, en el otro lado, la visualización del contenido del menú seleccionado.

Entrada de texto

Imagen 2.8 – teclado virtual

La carencia de un teclado físico en los dispositivos que llevan iOS ha hecho que en la interfaz se haya incluido un teclado virtual para que sea utilizado sobre la pantalla multitáctil. Cuando el usuario toca un campo de texto aparece un teclado completo en la parte inferior de la pantalla (tanto

en posición vertical como horizontal del dispositivo). Además, a medida que se va introducción un texto, la interfaz va comprobando automáticamente la ortografía e intenta predecir cuál será la palabra que se va escribir de manera que el usuario puede obtenerla con mayor rapidez.

2.2.2. Teléfono

En los iPhones, una de sus principales funciones es la de realizar y recibir llamadas telefónicas, además de otros servicios relacionados como sms, identificación de llamada, llamada en espera, etc. Otra función similar a la llamada telefónica es el FaceTime, servicio que permite realizar videoconferencias a otros dispositivos iOS a través de una red Wi-Fi con acceso a internet. El FaceTime requiere un requisito que solo cumple la última generación del iPhone, iPod Touch y iPad, tener una cámara frontal que permita enviar la imagen de video a los demás dispositivos.

Tanto la función de llamada telefónica como el FaceTime están totalmente integrados al iOS de forma que su utilización sea muy sencilla. La principal característica de esta integración es la de interrumpir de una aplicación que está en uso poniéndolo en suspensión o cerrándolo permitiendo atender la llamada. Una vez acabada la llamada el sistema operativo vuelve a reanudar o reiniciar la aplicación automáticamente.

2.2.3. Aplicaciones

Las aplicaciones son la piedra angular para todos los “smartphones”, a partir de ellas se define cuales son las funcionalidades que puede realizar éstos. En el caso del iOS incluye de forma predeterminada el siguiente listado de aplicaciones que cubren las necesidades básicas del usuario:

- Teléfono: recibir y enviar llamadas.
- Mail: cliente de correo electrónico.
- Safari: navegador web.
- iPod: reproductor multimedia iPod.
- Mensajes: recibir y enviar SMS y MMS.
- Calendario: gestor de citas y fechas especiales.
- Fotos: visor y gestor de fotos.
- Cámara: permite realizar fotos y videos.
- YouTube: visor de videos de YouTube.
- Bolsa: visor de cotizaciones de la bolsa.
- Mapas: visor del Google Maps.

- Tiempo: visor de temperaturas y clima en las ciudades deseadas.
- Reloj: Reloj mundial, alarma cronómetro y temporizador.
- Calculadora: calculadora normal y científica.
- Notas: bloc de notas rápidas.
- Notas de voz: grabadora de notas de voz.
- Ajustes: esta aplicación ajusta toda la configuración del dispositivo y realiza otras funciones relacionadas como por ejemplo conectarse a una red Wi-Fi, activación del Bluetooth, etc.
- iTunes Store: tienda de compra de música y videos de Apple.
- App Store: tienda de compra y descarga de aplicaciones.
- Brújula: realiza funciones de brújula.
- Contactos: gestor de datos (teléfono, dirección, correo electrónico, etc.) de personas u organizaciones.

multitasking

El multitasking o, en castellano, multitarea es la característica de ejecutar una aplicación en segundo plano. Cuando el usuario sale de una aplicación, el iOS guarda su estado en la memoria y lo deja activo, en caso de necesitar la ejecución de otras tareas o recibir eventos, con el mínimo de recursos posibles. De esta manera el usuario puede ejecutar otra aplicación y, cuando él quiera, volver a la primera aplicación en el mismo estado en que lo había dejado (o con datos actualizados en caso que estuviera ejecutado alguna tarea en segundo plano).

Esta característica fue introducida en el iOS 4, aunque el iPhone 3G no lo tiene activo debido a que no tiene suficiente memoria RAM para usarlo.

2.2.4. App Store

Si antes se comentó que las aplicaciones son la piedra angular para todos los “smartphones”, en el caso de Apple no es del todo exacto. Lo que realmente revolucionó fue la introducción de la aplicación App Store, servicio que permite al usuario comprar y descargar aplicaciones desde el mismo dispositivo. Gracias al App Store, el usuario puede obtener nuevas aplicaciones de una manera rápida y sencilla, y ampliando así el repertorio de usos que se le puede dar al dispositivo. El servicio no fue lanzado en la primera versión del iOS, sino que no salió hasta un año después en la segunda versión del sistema operativo. Así que antes de la segunda versión de iOS no había ningún método oficial para obtener nuevas aplicaciones.

El App Store está divida en cinco secciones o pestañas: destacados, categorías, top 25, buscar y actualizar. En destacados se puede ver aquellas aplicaciones que comienza a ganar fama por el aumento de descargas. En categorías están todas las aplicaciones dividas en diferentes familias según su enfoque, por ejemplo hay juegos, utilidades, música, viajes, etc. En top 25 se puede ver aquellas aplicaciones que han sido más descargada. En buscar permite realizar búsquedas de alguna aplicación en concreto. Y en actualizar muestra un listado de aplicaciones ya descargadas los cuales existe una nueva versión y permite actualizarlo. En el App Store se puede encontrar dos tipos de aplicaciones, aquellas que son de pago y las que son gratuitas, y en cada sección dividen los grupos de aplicaciones en estos dos tipos.

2.2.5. Actualizaciones

Tal como suele pasar en la mayoría de sistemas operativos, éstos reciben actualizaciones a medida que va pasando el tiempo. Normalmente estas actualizaciones van destinadas a arreglar errores, aunque otras veces son para introducir nuevas mejoras y funcionalidades. Por norma general, Apple libera una gran actualización (iOS x.0) durante la celebración anual del WWDC junto con la presentación del nuevo iPhone. Actualmente (2012), la última actualización es el iOS 5, presentado en el WWDC 2011.

2.3. iOS SDK

El iOS SDK es el Kit de Desarrollo de Software (traducción de Software Development Kit) desarrollado por Apple para los dispositivos con iOS. Al igual que el nombre del sistema operativo, anteriormente el kit se llamaba iPhone SDK, pero a partir de la salida del iOS 4 se cambió por el nombre actual.

El objetivo de este kit es permitir a terceros desarrollar aplicaciones nativas para los dispositivos iOS y aplicaciones web. Las aplicaciones nativas que se pueden desarrollar solo serán del tipo que se muestra en la pantalla principal (Home). No se puede desarrollar otro tipo de código como drivers o librerías dinámicas. No obstante, se puede utilizar el código de frameworks o librerías si están enlazadas estáticamente, en otras palabras, el código de éstos debe estar dentro del archivo ejecutable de la aplicación cuando éste sea creado. Las aplicaciones web son desarrollas utilizando la combinación de HTML, CSS y Javascript para que sean instaladas en servidores web y sean ejecutadas, a través de una red, desde el navegador de un dispositivo con iOS. La principal diferencia entre los dos tipos de aplicaciones es que la nativa necesita ser instalada en el dispositivo mientras que la web no tiene necesidad de ser instalada, simplemente se debe abrir el navegador y acceder directamente a la aplicación.

2.3.1. Historia

La primera versión beta fue liberada el 6 de marzo de 2008 y la primera versión oficial fue liberada el 11 de julio del 2008 junto con el lanzamiento del App Store.

Anteriormente a la liberación del SDK, Apple solo dispuso a los desarrolladores la ejecución de código a través del navegador web y utilizando AJAX, pero los resultados que se obtenían eran muy limitados. Debido a ésta y otros tipos de limitaciones, a mediados de octubre del 2007, un grupo de hackers consiguieron mediante el proceso denominado "jailbreak" poder realizar modificaciones en el código del iOS y permitir instalar aplicaciones no permitidas por Apple. El "jailbreak" aprovecha agujeros en el código de seguridad del iOS para poder acceder al núcleo del sistema y realizar los cambios oportunos para conseguir, además de instalar cualquier aplicación, nuevas funcionalidades, personalizaciones, etc. Actualmente, el "jailbreak" sigue funcionando y permitiendo instalar las aplicaciones que no están en el App Store, a pesar que Apple continuamente va tapado los diferentes agujeros de seguridad que utilizan los hackers.

La primera versión oficial del SDK correspondía al desarrollo del iOS 2.0. En cada liberación de la última actualización del iOS también liberan su correspondiente actualización del SDK, añadiendo nuevos APIs que dan acceso a las nuevas funcionalidades del nuevo iOS y capacidades de los nuevos dispositivos. Actualmente (2012), la última actualización del SDK se corresponde al iOS 5.

2.3.2. Requisito

El requisito necesario para poder comenzar a desarrollar con el iOS SDK es, además de la necesidad de tener conocimientos de programación y desarrollo del software, tener un equipo Mac con procesador Intel y Mac OS X. No hay más requisitos aparte de estos, debido a que el acceso del SDK es totalmente libre y gratuito, y para desarrollar no es necesario ser poseedor de un dispositivo iOS, ya que en el SDK contiene un simulador. Como dato concreto, hay que mencionar que en cada nueva actualización del iOS requiere que el Mac OS X disponga de la última actualización del sistema operativo.

Hay que remarcar que el SDK no está disponible para Microsoft Windows 7 ni para otro sistema operativo. Esto obliga a que todo desarrollador que tenga intenciones de trabajar con este SDK y que no tenga un equipo Mac tenga que adquirir este equipo de alguna manera. Esto ha suscitado muchas quejas por parte de los nuevos desarrolladores, ya que se han encontrado en la situación de tener que adquirir un segundo equipo.

Este modelo de liberación del SDK se está poniendo muy de moda entre las grandes empresas, a lo opuesto que se hacía antes, un previo pago de una licencia para poder comenzar a desarrollar. Las grandes empresas consiguen que por un lado los estudiantes de ingenierías de informática den sus primeros pasos en el desarrollo en sus plataformas y, por el otro lado, intentan atraer a los desarrolladores ya consagrados para que utilicen sus plataformas.

2.3.3. Contenido del SDK

El iOS SDK es un paquete que contiene todos los componentes necesarios para poder desarrollar aplicaciones para iOS y también Mac OS X. Los principales componentes que contiene el SDK son los siguientes:

- Frameworks con todas las librerías dinámicas oficiales y otros recursos (cabeceras de archivos, imágenes, etc.) para trabajar sobre iOS.
- Herramientas de Xcode : Paquete de aplicaciones que contiene herramientas para poder desarrollar aplicaciones para iOS. Las principales aplicaciones que compone este paquete son las siguientes:
    - Xcode: es un IDE (Integrated Developmente Environment) que permite gestionar un proyecto de desarrollo de una aplicación. Entre sus principales características permite editar, compilar, ejecutar y debugar el código escrito. Esta es la aplicación principal que se usa durante el desarrollo.
    - Interface builder: herramienta que permite configurar la vista de la interfaz de usuario.
    - Instruments: analizador del rendimiento en tiempo de ejecución de la aplicación y una herramienta para debugar. Gracias a esta herramienta se puede identificar los problemas de rendimiento como por ejemplo la pérdida de memoria.
- Simulador de iOS: que simula un dispositivo con iOS, más concretamente a un iPhone. Con el simulador se podrá comprobar el funcionamiento de la aplicación desarrollada.
- Biblioteca para desarrolladores de iOS: aplicación es la que el desarrollador puede encontrar documentación referencial y conceptual que enseña todo sobre la tecnología iOS.

2.3.4. Distribución

Una vez terminado el desarrollo de una aplicación existen tres maneras de distribuirlo: a través del App Store, por ad-hoc y a través de una distribución a nivel de empresa. Tanto para estas tres modalidades de distribución como para poder debugar con un dispositivo real con iOS, es necesario inscribirse al iOS Developer Program, con un previo pago. Existen diversas modalidades de este programa de desarrolladores, hay la versión gratuita para universidades que solo permite debugar con dispositivos reales; después existen la versión a nivel individual y otro a nivel de compañía que con un previo pago de 99 dólares al año permite debugar, distribución en el App Store y distribución por ad-hoc; y la última versión, la empresarial, con un pago de 299 dólares al año permite solo realizar la distribución a nivel de empresa (sin posibilidad de ad-hoc ni App Store).

La distribución por ad-hoc permite distribuir una aplicación hasta un máximo de 100 dispositivos a través de un email o colgándolo en un servidor web. En cambio, la distribución a nivel

de empresarial, creado para poder distribuir a todos los empleados las aplicaciones creadas para la empresa, no tiene ningún límite en cuanto al número de dispositivos en que se puede instalar.

2.3.5. App Store

Tal como se expuso en apartados anteriores, el App Store es la piedra angular del éxito de la plataforma de Apple y la principal forma de distribución de las aplicaciones desarrolladas. El éxito de este servicio viene, en parte, por la facilidad que tienen los usuarios para acceder a las nuevas aplicaciones y, por otra parte, por el modelo de negocio que se ha generado para los desarrolladores.

El modelo de negocio que propone Apple para las aplicaciones de pago es la siguiente: el 70 % de las ganancias por la venta de la aplicación son para el desarrollador y el 30 % restante para Apple. Apple argumenta que ese 30 % va destinado al mantenimiento del servicio del App Store. Naturalmente este modelo no es aplicable a las aplicaciones que son gratuitas al no generar ningún tipo de ganancia económica.

2.4. Cocos2D

Cocos2d es un framework que está especializado en la creación de gráficos, enfocado básicamente en la creación de videojuegos, aunque también se puede utilizar para la creación de cualquier otro tipo de aplicación. Un aspecto remarcable de este framework es el hecho de ser de código libre y que, por lo tanto, no requiere ningún tipo de pago por su uso.

Originalmente Cocos2d fue escrito en Python para crear juegos de dos dimensiones que fueran ejecutados en computadoras con sistemas operativos Windows, Linux y Mac OS X. En junio de 2008 se liberó la primera versión (la 0.1) de una nueva variante totalmente dedicada al desarrollo en la plataforma iOS. Esta nueva variante, llamada Coscos2d for iPhone, fue totalmente reescrita en el lenguaje de programa que se utiliza en las plataformas de Apple, Objective-C. El uso de este framework fue totalmente aceptado por Apple debido a que está licenciado con la MIT License. Durante los siguientes años, la comunidad que hace uso de Cocos2d for iPhone ha ido creciendo y han llegado a colgar en App Store 2500 aplicaciones que han sido creadas con este framework. Actualmente Cocos2d for iPhone está en la versión 1.0.

Cocos2d for iPhone contiene las siguientes características:

- Gestión de escenas.
- Transición entre escenas.
- Sprites y Sprite Sheets.
- Efectos: lente, onda, olas, liquido, giro, etc.
- Acciones:

  - Transformaciones.
  - Acciones compuestas.
  - Acciones de suavidad.
  - Otras acciones.
- Menús y botones básicos.
- Motor de física.
- Sistema de partículas.
- Renderización de textos.
- Texture Atlas.
- Tile Maps:
  - Mapas ortogonales.
  - Mapas isométricos.
  - Mapas hexagonales.
- Parallax scrolling.
- Sonido.
- Gestos y acelerómetro.
- Etc.

2.5. Otras herramientas

Además de las herramientas que se han ido enumerado en los anterior apartados, existen otras herramientas que también son necesarias para la ejecución del proyecto final de carrera las cuales su implicación en dicho proyecto es menor.

2.5.1. Mac OS X

Mac OS X es el sistema operativo desarrollado por la empresa Apple para ser instalada en las computadoras Macintosh de la misma empresa. Este sistema operativo está basado en UNIX y fue construido utilizando la tecnología de NeXT, empresa que fue absorbida por Apple.

La primera versión de Mac OS X (10.0.0) fue comercializada el 24 de marzo de 2001. La última versión oficial del sistema operativo es la 10.6.7 denominada Snow Leopard, aunque es

inminente la comercialización de la siguiente versión, la 10.7.0, denominada Lion. Las actualizaciones del sistema operativo son gratuitas al igual que iOS.

2.5.2. Gimp

Gimp es un editor de imágenes digitales. Algunas de las tareas que se puede realizar con esta aplicación es el retoque fotográfico, la composición de imágenes y la creación de imágenes, entre otras. Gimp es un acrónimo inglés formado por GNU Image Manipulation Program. Esta aplicación es libre y gratuita bajo la licencia GPL, y forma parte del proyecto GNU (proyecto que tiene por objetivo crear un sistema operativo completamente libre). Esta aplicación está disponible para muchos sistemas operativos, entre los que están Linux, Windows y Mac OS X, y en muchos idiomas, entre los que están el español, el catalán, el gallego y el euskera.

Las principales características de la aplicación son: creación de gráficos y logos, cambio de tamaño, recorte y modificación de fotografías digitales, modificación de colores, combinación de imágenes un paradigma de capas, eliminación o alteración de elementos no deseados o conversión a otros formatos de imagen.

Hoy en día, el Gimp se considera una alternativa libre y eficaz a la muy conocida aplicación Photoshop.

2.5.3. Dropbox

Dropbox es una aplicación que permite acceder al servicio de alojamiento de archivos en la nube de la compañía Dropbox. Este servicio permite al usuario almacenar y sincronizar archivos entre diversas computadoras. Dropbox se comercializa en dos versiones, una gratuita y otra de pago. Esta aplicación está disponible para sistemas operativos de computadoras como Windows, Mac OS X y Linux, y sistemas operativos de Smartphone como iOS, Blackberry y Android.

3. DESARROLLO EN EL IOS

En el capítulo anterior se han enumerado las herramientas necesarias para realizar este proyecto final de carrera, con su correspondiente análisis. Pero con solo conocer estas herramientas no se puede comenzar a trabajar, es necesario conocer y establecer una serie de conceptos de desarrollo de software. Estos conceptos se irán explicando en este capítulo y en los siguientes, a medida que sean necesario.

En este capítulo se explicaran un conjunto de conceptos básicos de desarrollo establecidos por Apple para poder trabajar a través del iOS SDK[9]. Este conjunto en realidad es simplemente un subconjunto de conceptos básicos, unas grandes pinceladas por así decirlo de cómo se debe trabajar. La razón por la que no se abarcan todos los conceptos básicos es que son tantos como para escribir un libro entero y, por lo tanto, solo se expondrán aquellos que sean más imprescindibles. Otro tipo de contenido que tampoco aparecerán será cómo utilizar las herramientas proporcionadas por el SDK, por las misma razones comentas anteriormente, y que en el capítulo anterior se puede encontrar una breve descripción.

El contenido de este capítulo se divide en los siguientes apartados: Objective-c, Arquitectura del iOS, Arquitectura de una aplicación, Ciclo de vida de una aplicación y Control táctil.

3.1. Objective-c

Objective-c es el lenguaje de programación orientado a objetos utilizados en las diferentes plataformas de Apple, tanto en Mac OS X como en iOS. Este lenguaje es un superconjunto del lenguaje de programación C que implementa un sistema de orientación a objetos inspirado en otro lenguaje llamado Smalltalk. Objective-c, debido a que es una pequeña capa que funciona sobre C, permite compilar cualquier programa escrito en C y, por lo tanto, podemos mezclar los dos lenguajes en una clase de Objective-c.

3.1.1. Archivos

Al igual que pasa con el lenguaje C, los archivos que contiene todo el código está dividido en dos tipos de archivos, el archivo de cabecera y el archivo del código fuente. Los archivos de cabecera contienen la declaración de las clases, tipos, funciones, etc… y utiliza la extensión “.h”. Y los archivos de código fuente contienen todo el código de implementación de las clases y las funciones han sido declaradas en el archivo de cabecera, y la extensión que utiliza el archivo es el “.m”. En el caso de querer utilizar un archivo de cabecera se utilizará la directiva #import que funciona igual que la directiva #include de C.

[9] Software Development Kit o, traducido al español, kit de desarrollo de software.

3.1.2. Clases

Las clases en Objective-c están compuestas por dos partes muy diferenciadas: la interfaz y la implementación. La parte de la interfaz contiene la declaración de la clase, los atributos y los métodos asociados que la definen. Normalmente, esta parte se suele escribir en el archivo con la extensión ".h". La parte de implementación contiene el código de los métodos declarados en la otra parte. Normalmente, la implementación se escribe en el archivo con la extensión ".m".

En la imagen 3.1 se puede ver la sintaxis de la declaración de la clase MyClass, el cual hereda de la clase base de Cocoa[10], el NSObject. La declaración de la clase comienza con la directiva @interface y acaba con la directiva @end. A continuación, en la directiva @interface encontramos el nombre de la clase y, separando con dos puntos, el nombre de la superclase que hereda. Después, se encuentra entre dos llaves el bloque de la declaración de los atributos de la clase que en este ejemplo son tres variables. Y el último bloque es la declaración del listado de métodos.

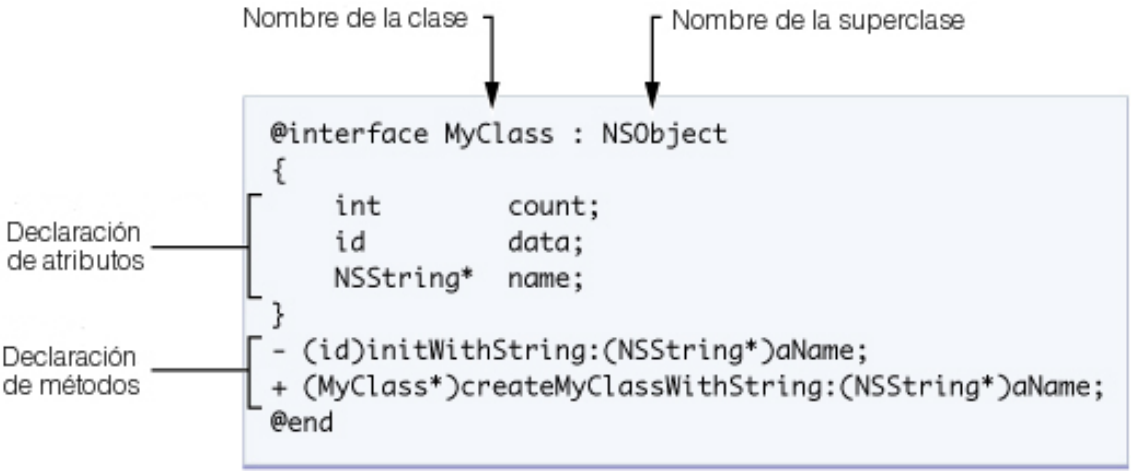

Imagen 3.1

Objective-c soporta los dos tipos de tipeados para referenciar una variables de tipo objeto. Por una parte, el tipeado débil, que está indicado por el uso del tipo id , permite hacer referencia a un objeto sin tener que declarar a que tipo de clase pertenece. Y por otro lado, el tipeado fuerte, que está indicado por el uso del nombre de la clase del objeto y precedido del signo * que indica que la variable es un puntero. Hay que indicar que el tipo id actúa también como un puntero a un objeto.

A continuación se va mostrar unos ejemplos de los tipos de tipeados:

```
MyClass *myObject1;      // Tipeado fuerte

id          myObject2;   // Tipeado débil
```

3.1.3. Métodos y mensajes

Una clase en Objective-c puede declarar dos tipos de métodos: métodos de instancia y métodos de clase. Los métodos de instancia son métodos que son ejecutados por una instancia de

[10] En el apartado de Arquitectura del iOS se hablará de Cocoa.

aquella clase y, por lo tanto, se necesita crear una instancia de esa clase antes de ser ejecutados. En cambio, los métodos de clase no necesitan de este requerimiento para ser ejecutados.

En la imagen 3.2 se puede ver la sintaxis de la declaración de un método, concretamente la declaración del método de instancia insertObject:atIndex: .

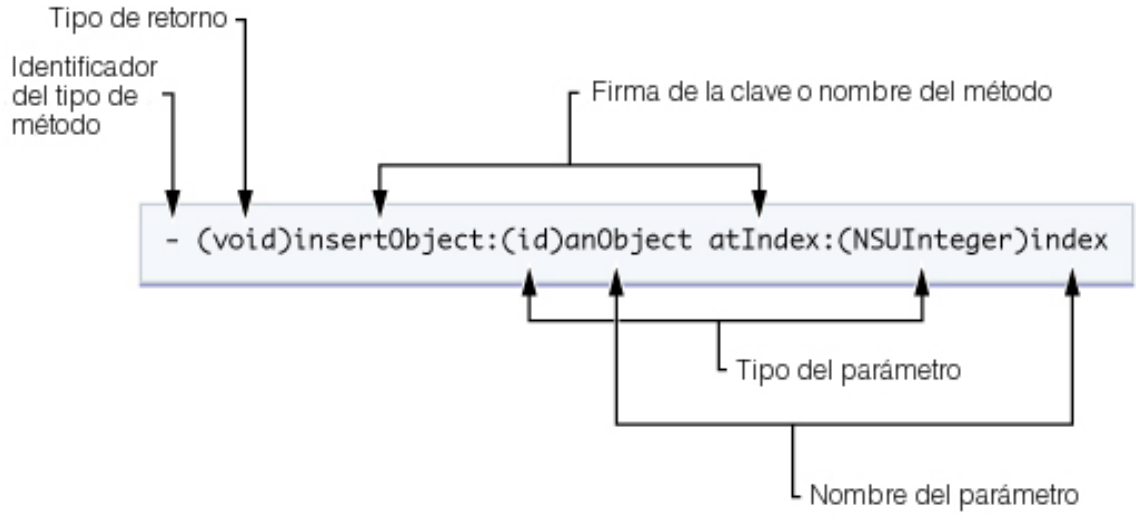

Imagen 3.2

La declaración del método comienza con el signo – que indica que el método es un método de instancia. El nombre del método esta compuesto por la concatenación del nombre del método y de los parámetros más el signo :, por ejemplo el método de la imagen 3.2 (insertObject:atIndex:) contiene dos signos de : y, por lo tanto, quiere decir que tiene dos parámetros. En el caso que un método no tuviera parámetros se puede olvidar de añadir el signo de :.

Cuando se necesite realizar una llamada a un método se deberá enviar un mensaje a su correspondiente objeto. El envío de un mensaje se efectúa colocándo dentro de dos corchetes ([y]) el objeto —a la izquierda— y el nombre del método —a la derecha— (y también las variables de los parámetros en caso que el método los tenga). A continuación se mostrará la llamada al método insertObject:atIndex:).

```
[myArray insertObject:anObject atIndex:0];
```

La sintaxis de los corchetes también acepta la combinación de más corchetes correspondientes a otros envíos de mensajes. En el siguiente ejemplo se mostrará una combinación de corchetes utilizando el objeto myAppobject que tiene métodos que devuelven un objeto:

```
[[myArrayObject theArray] insertObject:[myAppObject objectToInsert] atIndex:0];
```

Además de esta sintaxis también existe otra como la sintaxis del punto . que permite realizar llamadas a métodos de acceso a atributos de una clase. Los métodos de acceso a atributos permiten acceder y modificar el estado del atributo. La declaración de estos métodos son - (type) PropertyName para acceder y – (void) setPropertyName:(type) para modificar. Esta sintaxis también

se puede combinar con la sintaxis de corchetes. A continuación se muestra un ejemplo de la misma llamada de un método para las dos sintaxis:

```
myAppObject.theArray = aNewArray;          // sintaxis del punto .
[myAppObject setTheArray:aNewArray];       // sintaxis de los corchetes
```

Anteriormente, en la imagen 3.2, se habló que el signo –, que estaba al inicio de la declaración de un método, indicaba que era un método de instancia. En el caso de querer utilizar métodos de clase, por ejemplo para utilizarlo en un método del patrón factory para crear nuevas instancias de esa clase, se debe utilizar el signo +. En el siguiente ejemplo se puede ver como se crea un objeto de la clase NSMutableArray utilizando su correspondiente método del patrón factory:

```
NSMutableArray *myArray = nil;      // nil es lo mismo que NULL
myArray = [NSMutableArray array]; // creando y asignado un nuevo array
```

A continuación se muestra la segunda parte del código de una clase en Objective-c, la implementación para el ejemplo de la clase MyClass.

```
@implementation MyClass

- (id)initWithString:(NSString *)aName
{
    self = [super init];
    if (self) {
        name = [aName copy];
    }
    return self;
}

+ (MyClass *)createMyClassWithString: (NSString *)aName
{
    return [[[self alloc] initWithString:aName] autorelease];
}
@end
```

Esta parte del código está definida por la directiva @implemetation y acaba con @end. Esta directiva proporciona al compilador la asociación de la declaración de los métodos en @interface con la implementación de ellos.

3.1.4. Declaración de propiedades

Las propiedades son un método de declaración, que utilizando la directiva @property, permite a un atributo de una clase ahorrase la declaración e implementación de métodos de acceso a el. La directiva @property tiene una serie de opciones que permite configurar que tipo de comportamiento tendrá esos métodos. La convención para utilizar los métodos creados por la propiedad son los mismos que se comentó en el párrafo de la sintaxis del ".". A continuación se muestra una serie de ejemplos:

```
@property BOOL flag;
```

```
@property (copy) NSString *nameObject;  // Copia el objeto durante su asignación
@property (readonly) UIView *rootView;  // Declara solo un método solo para leer
```

Para completar el proceso de obtención de estos métodos de las propiedades se debe escribir la directiva @synthesize en la parte de implementación de la clase, tal como aparece en el siguiente ejemplo.

```
@synthesize flag;
@synthesize nameObject;
@synthesize rootView;
```

3.1.5. Protocolos

Un protocolo es una forma de declaración de métodos que pueden ser implementados por cualquier clase. Los protocolos no son clases, simplemente definen una interfaz que otro objeto tiene la responsabilidad de implementarlo. El uso habitual de los protocolos es para declarar una interfaz para que los utilice los objetos de delegados (patrón delegación). En el siguiente ejemplo se muestra como se debe utilizar un protocolo:

```
@interface MyClass : NSObject <UIApplicationDelegate, AnotherProtocol> {
}
@end
```

Tal como se puede ver, los protocolos son declarados a continuación del nombre de la superclase que hereda la clase y está colocado entre los signos < y > . A continuación se muestra un ejemplo de cómo se debe declarar un protocolo.

```
@protocol MyProtocol
- (void)myProtocolMethod;
@end
```

Se puede apreciar que la declaración del protocolo es muy similar a la interfaz de una clase.

3.2. Arquitectura del iOS

La arquitectura del sistema operativo iOS es muy similar a la arquitectura que podemos encontrar en el Mac OS X. Fundamentalmente, y tal como hacen todos los sistemas operativos, el iOS actúa de intermediario entre el conjunto de hardware del dispositivo y las aplicaciones aparecen en la pantalla. Las aplicaciones creadas normalmente no se comunican directamente con el hardware, sino que se comunican a través de un sistemas de interfaces. La misión de este sistema de interfaces es ofrecer a las aplicaciones protección de futuros cambios del hardware. Esta abstracción también ayuda a poder trabajar con diferentes hardwares.

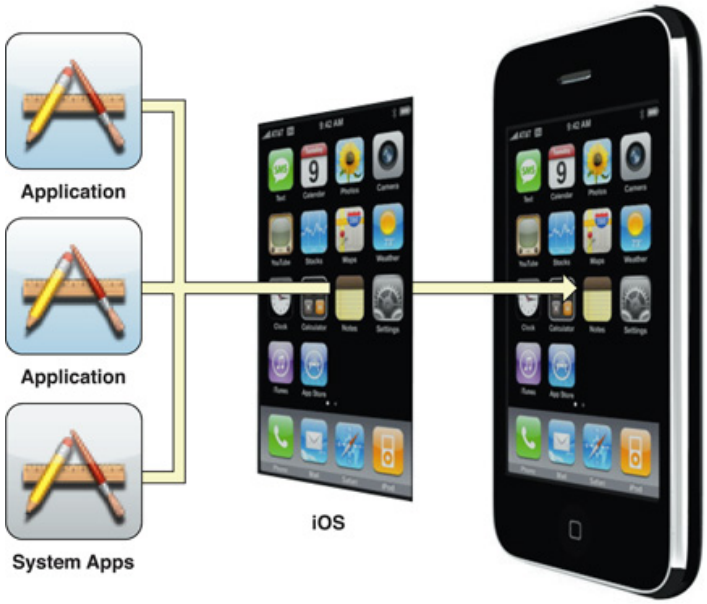

Imagen 3.3 – La capa de aplicaciones encima del iOS

La implementación de las tecnologías del iOS sigue el patrón arquitectónico en capas, tal como se puede ver la imagen 3.4. El concepto fundamental es dividir los diferentes componentes entre las diferentes capas, de forma que las capas más bajas son aquellas con servicios y tecnologías fundamentales en que se basan todas las aplicaciones.

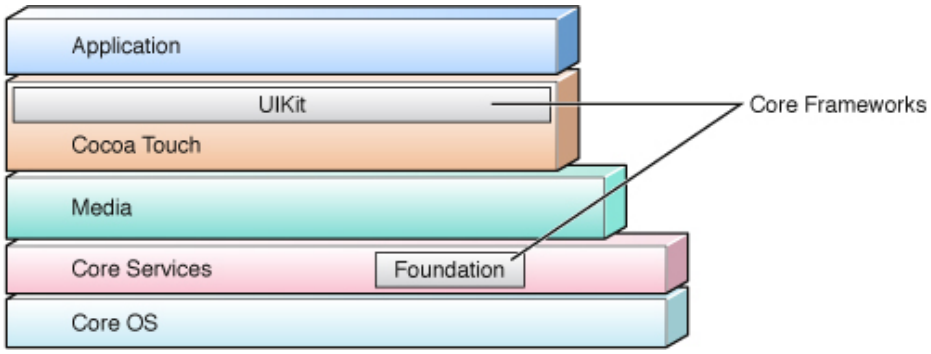

Imagen 3.4 – Capas del iOS

A la hora de escribir el código se debe usar antes los frameworks de las capas de más alto nivel que las de bajo nivel tanto como sea posible. Los frameworks que están en las capas de más alto nivel son aquellos que suministran la mayoría de características más usadas para el desarrollo de aplicaciones. Estos frameworks ahorran mucho trabajo sucio en la implementación de tareas de bajo nivel, permitiendo hacer muchas cosas con pocas líneas de código. Aún así, los frameworks de las capas de más bajo nivel están accesibles para utilizarlas para tareas que no se pueden hacer en los frameworks de alto nivel.

3.2.1. Cocoa Touch

Tal como se ha comentado anteriormente, la capa de Cocoa touch contiene los frameworks clave para el desarrollo de aplicaciones. Esta capa define la estructura básica para las aplicaciones y da soporte a tecnologías claves como la multitarea, la entrada de toques de las pantalla, las notificaciones, etc. El framework más usado e importante en esta capa es el UIKit.

3.2.2. Media Layer

La capa de media layer suministra los frameworks que gestiona las tecnologías de gráficos (2D y 3D), audio y video. En otras palabras, esta capa se encarga de dar acceso al potencial multimedia del hardware.

3.2.3. Core Services Layer

La capa del núcleo de servicios es una capa que ya se considera de bajo nivel y que proviene de características como el control de concurrencia de hilos, las compras de contenidos en las aplicaciones, SQLite (motor ligero de base de datos), y el soporte a xml, Los servicios que provienen esta capa son acceso a la información de aplicaciones básicas de iOS como agenda de teléfonos, el teléfono, localización y ajustes, servicios de comunicación a redes (como internet), manipulación de strings, gestión de colecciones de objetos, etc. Los frameworks más importantes de esta capa son el Foundation y el Core Foundation.

3.2.4. Core OS Layer

La capa del núcleo del sistema operativo controla las características de bajo nivel que las demás tecnologías y frameworks se implementan sobre ellas. Los pocos frameworks que se pueden acceder en esta capa son aquellos que tratan sobre la aceleración de cálculos matemáticos, accesorios externos y seguridad. Además de estos framework, esta capa también contiene el kernel (núcleo) del sistema operativo y otros componentes que actúan sobre el kernel. El acceso al kernel y sus otros componentes son en gran parte restringidos por razones de seguridad.

Los frameworks más importantes en el desarrollo de aplicaciones para iOS, los denominados “core Cocoa Frameworks”, son el UIKit y el Foundation.

- UIKit. Este framework contiene los objetos que la aplicación muestra en la interfaz de usuario y define la estructura del comportamiento de la aplicación, incluyendo el control de los eventos y el dibujado de la pantalla.
- Foundation. Este framework define el comportamiento básico de los objetos estableciendo mecanismos para su gestión, y conteniendo objetos de tipo de datos primitivos, colecciones, y servicios del sistema operativo.

3.3. Arquitectura de una aplicación

En el apartado anterior se ha visto que el patrón de diseño que utiliza la arquitectura del iOS es el de capas, pues en el caso de la arquitectura de una aplicación se sigue más o menos la misma filosofía pero con un patrón derivado de este, el MVC.

3.3.1. Patrón Modelo Vista Controlador

El patrón de diseño Modelo Vista Controlador (MVC) consiste en repartir todos los objetos entre tres grupos o capas, el colectivo de objetos de cada grupo desempeñará entre todos un rol de entre estos tres: modelo, vista o controlador. Este patrón define los limites abstractos entre los objetos de las diferentes capas y una determinada forma de comunicación entre ellos. Las ventajas en adoptar este patrón de diseño son: la reusabilidad de muchos de los objetos debido a que sus interfaces están bien definidas, y la escalabilidad del sistema que permite extenderse con nuevas funcionalidades en la aplicación.

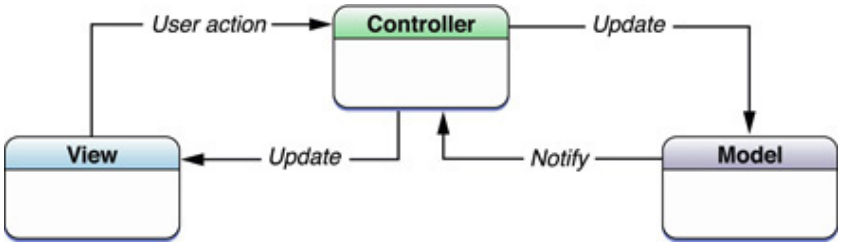

Imagen 3.5 – Patrón Modelo Vista Controlador

- Capa del Modelo (Model): Los objetos de la capa del modelo encapsulan los datos específicos de la aplicación y definen la lógica de los procesos que manipulan esos datos. Esta capa también es responsable de la persistencia de los datos, permitiendo cargar el estado de los datos (por ejemplo desde un fichero o una base de datos) cada vez que se inicializa la aplicación. Debido a que los objetos del modelo representan el conocimiento y están relacionados con el dominio de un problema, se pueden reusar los mismos objetos en otros dominios de problemas similares. El ideal de esta capa es que sus objetos no tengan una relación explícita o directa con los objetos de la capa vista y a su forma representar los datos a la interfaz de usuario.

- Capa de la Vista (View): Los objetos de esta capa son aquellos objetos que el usuario puede ver. Un objeto de la capa de la vista ha de saber dibujarse y poder responder a las acciones del usuario. El mayor propósito de esta capa es mostrar los datos de la aplicación (ubicados en la capa del modelo) y permitir la edición de estos datos. A pesar de todo, todos los objetos de la capa de la vista normalmente están desacoplados de los objetos de la capa del modelo. Debido a este desacoplamiento, esta capa también puede reutilizar y modificar sus objetos sin que afecte a la capa del modelo.

- Capa del Controlador (Controller): Los objetos de la capa del controlador actúan de intermediario entre la capa de la vista y la capa del modelo. De esta manera, los objetos

del Controlador hacen de conductor permitiendo que los objetos de la capa de la vista tengan conocimiento de los cambios en los objetos de la capa de modelo y viceversa. Otras de las tareas que se encarga es configurar el rendimiento y coordinar las tareas de la aplicación, y gestionar el ciclo de vida de los objetos.

3.3.2. Objetos del núcleo de una aplicación

Desde el momento en que se lanza una aplicación y hasta que este finaliza, el framework UIKit gestiona muchos de los comportamientos de la aplicación. Un ejemplo del trabajo que se encarga la aplicación es recibir continuamente los eventos del sistema y responder a todos ellos. La recepción de los eventos es trabajo del objeto UIAplication, pero la responsabilidad de responderlos es del código personalizado por el programador de la aplicación. Este comportamiento tiene una relación similar en las demás partes de la aplicación, con objetos de sistema que gestionan sobre todos los procesos y que el código personalizado del programador enfoca en la implementación de un comportamiento específico de la aplicación.

Para entender el funcionamiento de los objetos del UIKit con el código personalizado del programador se muestra, en la imagen 3.6 un esquema de los objetos clave de una aplicación sobre el patrón MVC.

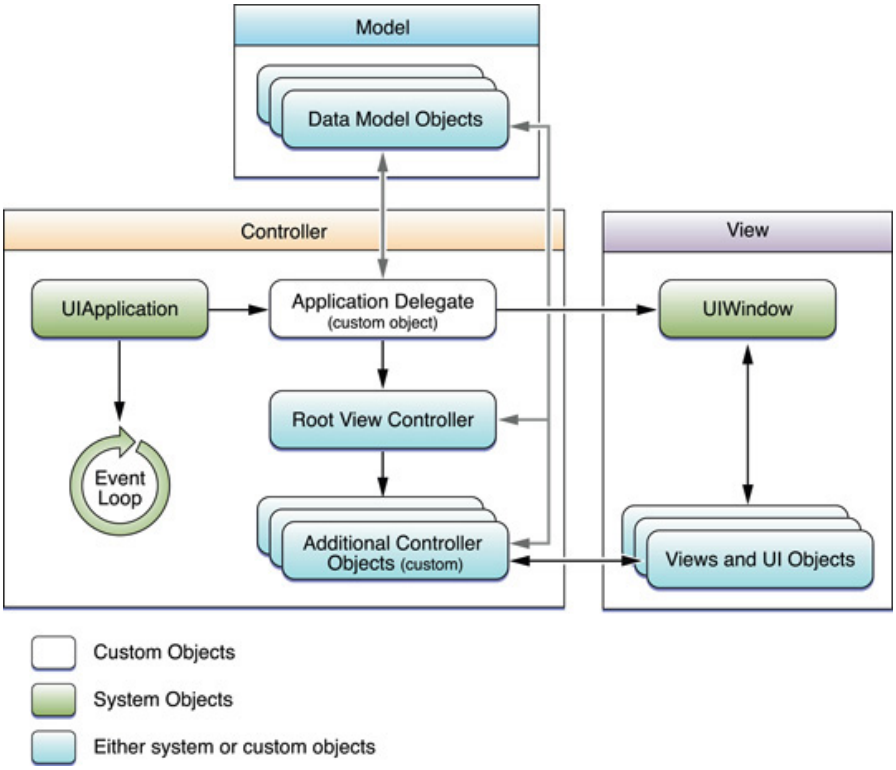

Imagen 3.6 – Objetos clave en una aplicación del iOS

La imagen 3.6 muestra en cada capa del patrón MVC una serie de objetos que pueden ser de tres tipos: los objetos de color blanco son aquellos que son totalmente personalizados con el comportamiento que el programador quiere dar a la aplicación, los objetos de color verdes son aquellos que son controlados por el sistema operativo y que el programador no puede realizar

ninguna modificación, y los objetos de color azul son una mezcla dos los anteriores, son objetos controlados en parte por el sistema operativo y pueden ser parcialmente modificados por el programador. En la siguiente lista se da una breve descripción del papel que tiene cada objeto clave de la aplicación:

- UIApplication: este objeto gestiona el bucle de eventos de la aplicación y coordina los otros comportamientos de alto nivel. El programador puede configurar varios aspectos de la aplicación a partir de este objeto. para realizarlo será necesario implementar el código en el objeto delegado del UIApplication, ya que ambos objetos trabajan juntos.

- Application delegate: el delegado de la aplicación es el objeto personalizable ofrecido al programador durante el inicio de la aplicación y que normalmente está embebido dentro del archivo ".nib" del main de la aplicación. La principal tarea de este objeto es inicializar la aplicación y presentar el objeto ventana en la pantalla. El objeto UIAplication también notifica al delegado cuando ocurre un evento específico, al igual que cuando la aplicación necesita ser interrumpida.

- Data model : Los objetos del modelo de datos almacenan el contenido de la aplicación y que, por lo tanto, son específico de la aplicación. Por ejemplo, en una aplicación de un banco se puede almacenar la base de datos de las transacciones.

- Controlador de la vista: Los objetos del controlador de la vista gestionan la presentación del contenido de la aplicación. Normalmente, esto concierne sobre la creación de vistas que presenten el contenido y la gestión de la interacción entre la vistas y los objetos del modelo de datos. La clase UIViewController es la clase base de todos los controladores de vista. Este ofrece funcionalidades como la animación en el cambio de vistas, recoger los eventos de rotaciones de la pantalla, y muchos otros comportamientos.

- UIWindow: Este objeto coordina la presentación de una o más vistas en la pantalla del dispositivo. Normalmente en las aplicaciones solo utilizan una ventana y esta contiene una o más vistas. En el caso que una aplicación cambiara de ventana también estaría cambiando todo el conjunto de vistas. El UIWindow, además de hospedar vistas, también es responsable de entregar los eventos de las vistas a sus respectivos controladores de vista.

- Vistas, controles y capas: Las vistas y los controles ofrecen la representación visual del contenido de una aplicación. Una vista es una objeto que dibuja el contenido en una área rectangular y responde a los eventos que surgen dentro de ella. Los controles son un tipo especial de vistas que son responsables de implementar objetos de interfaz tan familiares como por ejemplo botones, campos de texto o interruptores. Además de las incorporaciones de vistas y controles, una aplicación puede también incorporar capas. Los objetos de capas son realmente objetos con datos de la representación de un contenido visual. Un objeto capa personalizado puede por ejemplo implementar en una

interfaz de usuario una compleja animación y sofisticados efectos visuales.

3.4. Ciclo de vida de una aplicación

El ciclo de vida de una aplicación constituye una secuencia de eventos que ocurren entre el inicio y la finalización de la aplicación (en otras palabras, desde el momento en que el usuario toca el icono de la aplicación hasta que toca el botón del menú principal). Una vez iniciados los primeros métodos que ejecuta la aplicación, que son el main() y UIApplicationsMain(), carga el archivo tipo .nib mainWindow, a partir del framework UIKit. Una vez iniciados estos métodos, el UIKit se encarga de iniciar un bucle que recoge todos los eventos (Event Loop) que suceda en la pantalla.

En la imagen 3.7 se puede ver un esquema del ciclo de vida de una aplicación que acaba de ser iniciada. El esquema muestra un resumen de los eventos que van sucediendo. En el lado derecho se puede ver los mensajes que el UIKit va enviando al objeto delegado de la aplicación permitiendo saber cuando esta sucediendo estos eventos.

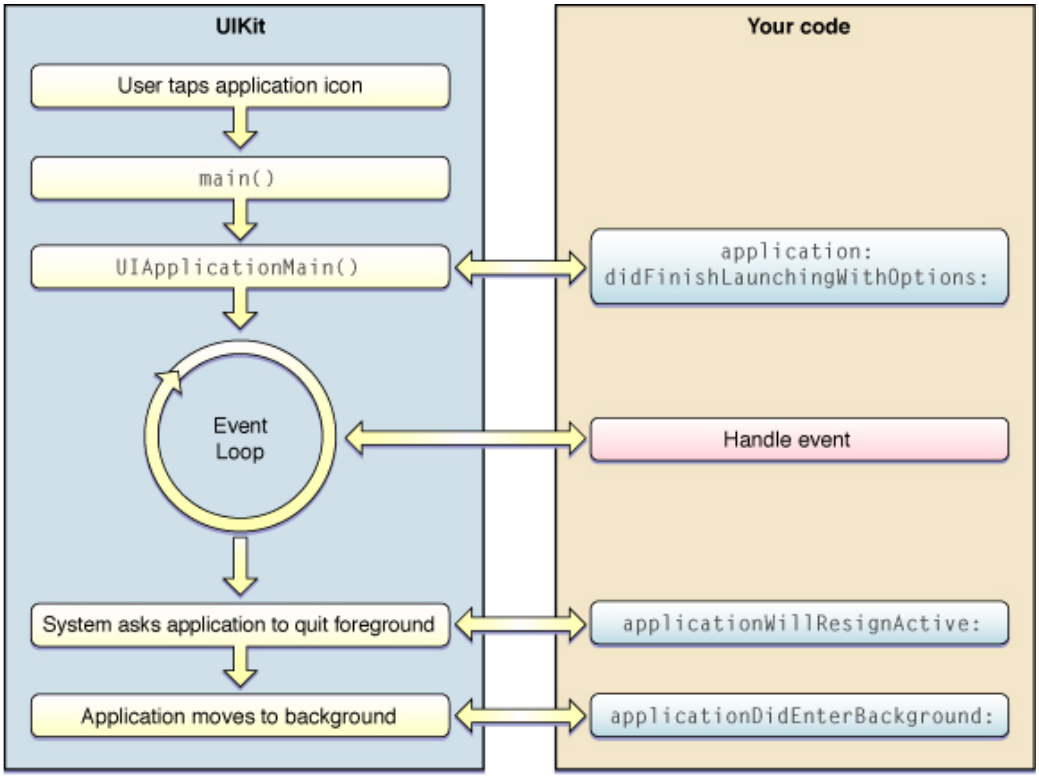

Imagen 3.7 – Ciclo de vida de una aplicación

A continuación se muestra una lista de los diferentes estados en los que puede permanecer una aplicación:

- Not running (sin ejecutar): La aplicación no ha sido iniciado o estuvo en ejecución pero fue finalizada.
- Inactive (Inactivo): La aplicación está en ejecución en primer plano pero no recibe eventos. La aplicación entra en este estado puente para ayudar en la transición entre diferentes

estados. Esto suele pasar cuando la pantalla es bloqueada o cuando llega el aviso de algún evento como una llamada entrante o cuando recibe un mensaje SMS.

- Active (Activo): La aplicación está en ejecución en primer plano y está recibiendo eventos.
- Background (Segundo plano): La aplicación está en segundo plano y ejecutando código. Muchas aplicaciones entran en este breve estado antes de pasar al estado de suspensión. En este estado la aplicación puede ejecutar un determinado tipo código durante un cierto tiempo.
- Suspended (Suspendido): La aplicación está en segundo plano pero no está ejecutando ningún código. Cuando la aplicación está suspendida esencialmente se congela en el mismo estado que estaba antes de suspenderse y sin ejecutar nada, así cuando vuelva estar en activo, continuará por el mismo punto de ejecución.

Tanto el esquema de la imagen 3.7 como el listado de estados de una aplicación dependen directamente del sistema multitarea del iOS 4. En el caso que el dispositivo tenga una versión más vieja del iOS o si no tiene capacidad para ejecutar el sistema multitarea, el esquema y la lista serán un poco diferentes, ya que los dos últimos estados no existirían (segundo plano y suspendido) y, en el esquema, la aplicación terminaría (estado sin ejecutar) en vez de pasar a segundo plano.

En los siguientes subapartados se van a mostrar los diferentes procesos que sufre una aplicación en su ciclo de vida, en los cuales se explicarán los sucesos que van pasando y los eventos que va recibiendo el objeto delegado de la aplicación.

3.4.1. Inicio de una aplicación

El proceso de inicio de una aplicación tiene como objetivo el cambio de estado inactivo, de una aplicación que aún no ha sido ejecutada a estado activo. A continuación se enumeran los sucesos y eventos que ocurren en esta transición, tal como aparece en la imagen 3.8:

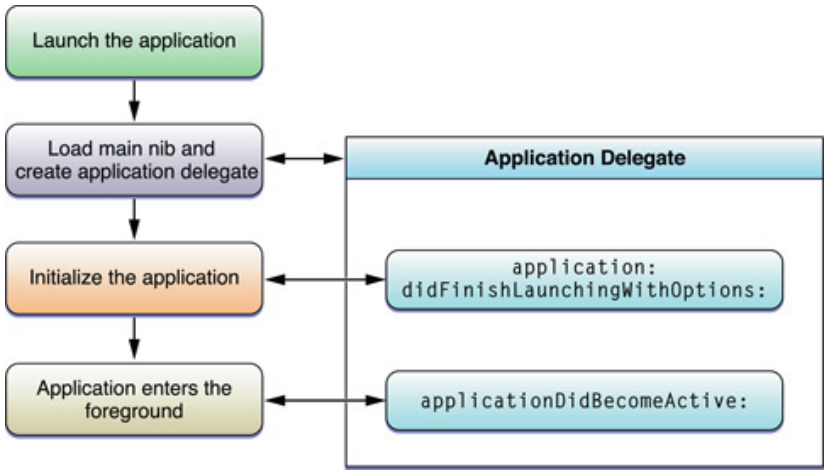

Imagen 3.8 – Inicio de una aplicación

- Inicio de la aplicación.
- Carga del archivo main de tipo .nib y creación del objeto delegado de la aplicación.

- Inicialización de los parámetros principales de la aplicación. El objeto delegado de la aplicación recibe el evento application:didFinishLaunchingWithOptions: indicando que se acabado de iniciar la aplicación y permitiendo al programador incluir el código para cargar la estructura de datos, configurar orientación de la pantalla, etc.
- La aplicación entra en primer plano y lo indica enviado el evento applicationDidBecomeActive: al objeto delegado de la aplicación. Este evento indica que la aplicación que acaba de pasar del estado inactivo al estado activo y, como los otros eventos, se puede incluir código personalizado.

3.4.2. Pasando a segundo plano

Una vez una aplicación está activa y en ejecución éste puede pasar al segundo plano por aceptar la notificación de una llamada entrante, por aceptar una notificación de un SMS o simplemente por apretar el botón del menú principal. A continuación se describe que eventos interviene en el proceso que se muestra en la imagen 3.9:

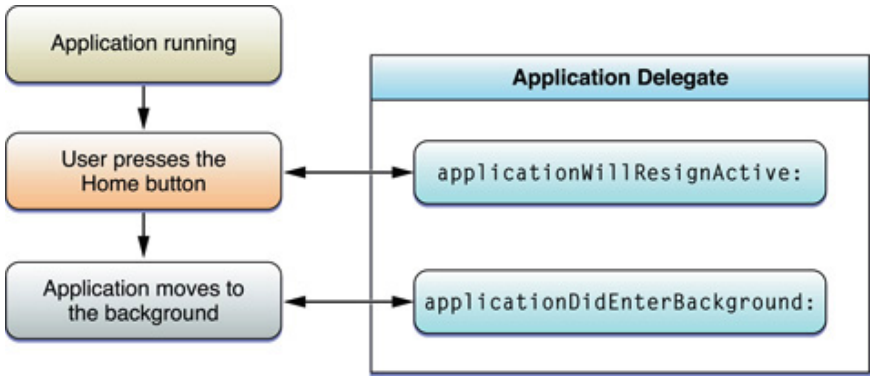

Imagen 3.9 – Paso de primer plano (activo) a segundo plano

- La aplicación está en ejecución.
- El usuario aprieta el botón de menú principal. Esto provoca que el objeto delegado de la aplicación reciba el evento applicationWillResignActive: indicando que la aplicación cambiará del estado activo a otro estado. La personalización del código que se suele escribir al recibir este evento consiste en guardar datos que están el cache, liberar de imágenes en el cache, entre otros.
- La aplicación cambia al estado de segundo plano y lo notifica enviando el evento applicationDidEnterBackground:.

3.4.3. Volviendo al primer plano

Estando aún la aplicación en segundo plano, éste puede volver al primer plano (activo) cuando quiera el usuario. El proceso que interviene en esta vuelta al primer plano es el siguiente:

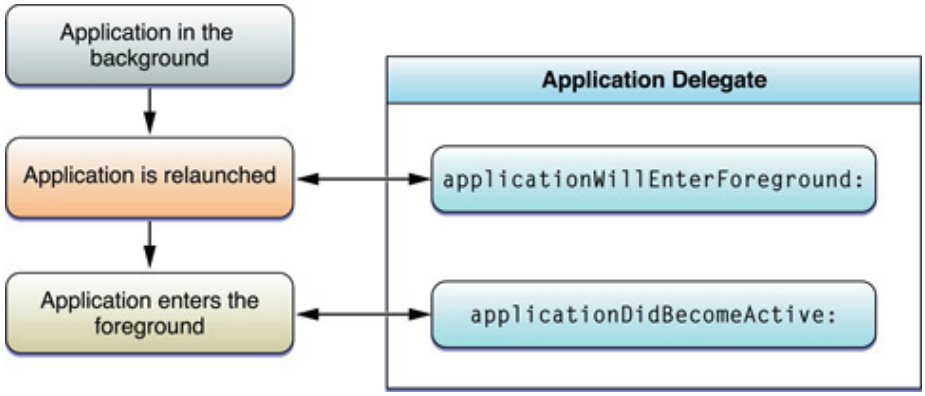

Imagen 3.10 – Volviendo al primer plano

- La aplicación está en segundo plano.
- La aplicación es reiniciada por el usuario y el objeto delegado recibe el evento applicationWillEnterForeground:. Este evento indica que cambiará al primer plano. En relación al código personalizado para este evento sería similar al del evento application:didFinishLaunchingWithOptions: de inicio de la aplicación.
- La aplicación acaba de cambiar de estado y vuelve a estar activo. Para indicar el cambio a activo el objeto delegado recibe el evento applicationDidBecomeActive:.

3.4.4. Finalización de una aplicación

Una aplicación es finalizada sin pasar al segundo plano y borrada de la memoria del dispositivo por la siguientes razones:

- La aplicación está implementada para iOS inferiores al 4.0.
- La aplicación se está ejecutando en un iOS inferior al 4.0.
- El dispositivo no soporta multitarea.
- La aplicación está configurada utilizando la clave UIApplicationExitsOnSuspend, en el archivo Info.plist.

Otro motivo de la finalización de la aplicación, tanto cuando está en primer como segundo plano, es por falta de memoria. Tanto por esta razón como por las de la lista anterior provocan recepción del evento applicationWillTerminate:, permitiendo personalizar código para guardar los datos de la aplicación que permitirán recuperarlos en el siguiente inicio de la aplicación.

3.5. Gestión de memoria

3.5.1. Introducción

En Objective-c existen dos maneras de gestionar la persistencia de los objetos, tanto cuando son necesarios como cuando son destruidos y, por lo tanto, son liberados de la memoria. Las dos maneras son las siguientes:

- Recolector de basura: esta característica de Objective-c es un sistema que gestiona automáticamente todos los objetos creados en tiempo de ejecución, realizando automáticamente la reserva de memoria y la destrucción del objeto sin que el programador tenga que preocuparse.
- Gestión de memoria: este método consiste en que el programador tiene el total control y responsabilidad de crear, inicializar y liberar los objetos. Utilizando los métodos adecuados en cada momento, el programador gestionará un contador que controla la vida del objeto.

Está claro que el recolector de basura supone un sistema cómodo y que ahorra una parte de código de la implementación de una aplicación, pero por desgracia este sistema no está soportado en el iOS. Sin el recolector de basura, la única manera de gestionar la persistencia en el iOS es mediante la gestión de memoria.

3.5.2. Gestión de memoria

La política que sigue la gestión de memoria es la que de un objeto se mantiene mientras este sea útil y que en el momento no haya más necesidad de él sea destruido. Para conseguir este objetivo se utiliza el concepto de contadores de referencia o retención. Este concepto se basa en que el contador es un número que indica cuantos objetos están interesados en su persistencia. En el momento que un objeto es creado este contador se inicia a uno y, a medida que otros objetos se interesan y dejan interesarse, el contador va subiendo y bajando. En el momento que el contador llega a cero, quiere decir que nadie le interesa ese objeto y, por lo tanto, se puede destruir.

Para facilitar la tarea de la gestión de memoria, Cocoa proporciona una clase base, NSObject, que tiene implementado el contador y los métodos para controlarlo. La condición que debe cumplir las clases que quieran el suporte de gestión de memoria ofrecido por Cocoa es heredar la clase NSObject. Por esta razón, es muy recomendable que todas las clases implementadas hereden de esta clase. En la siguiente lista se resume cuales son los principales métodos de la clase NSObject que gestiona la memoria:

- Alloc: este método se encarga de reservar memoria para un objeto e inicia el contador de referencia a 1.
- Init: este método se encarga de inicializar el objeto y sus atributos. El programador debe sobrescribir este método y personalizar la inicialización.
- Copy: este método devuelve un objeto que es una copia exacta del objeto. El contador de referencia del nuevo objeto estará a 1.
- Retain: este método hace aumentar en 1 el contador de referencia.
- Release: este método hace disminuir en 1 el contador de referencia.
- Autorelease: hace la misma función que el método release con la diferencia de que no lo hará inmediatamente sino que lo hará en el futuro.

- Dealloc: este método realiza la destrucción de los objetos tipo atributo de la clase. Este método debe ser sobrescrito y personalizado por el programador, de igual manera que el método init, aunque con la diferencia que este método no debe ser llamado por el programador sino que se autollama solo cuando el contador de referencia del objeto llega a 0.

En la imagen 3.11 se muestra un ejemplo de la utilización de varios de los métodos vistos anteriormente con su respectivo contador de referencia, permitiendo de esta manera poder más fácilmente el ciclo del objeto anObj.

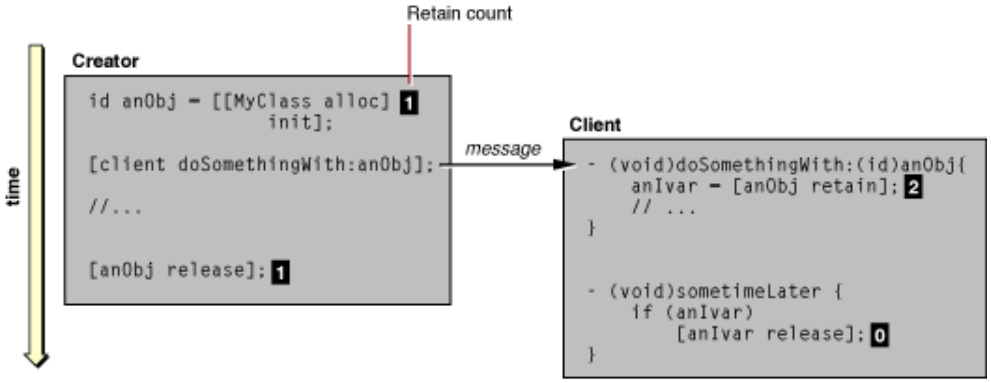

Imagen 3.11

De los métodos en listados anteriores, el autorelease es un método especial que necesita de otro objeto para funcionar, concretamente el NSAutoreleasePool. El NSAutoreleasePool consiste en un objeto, que una vez creado, guarda la referencia de los objetos que haya hecho la llamada autorelease. Todos los objetos que hayan hecho la llamada autorelease no disminuirán en 1 su contador de referencia hasta que el objeto anterior de la clase NSAutoreleasePool no sea destruido. La utilización del sistema de autorelease no es muy recomendado ya que tiene la desventaja de que si hay muchos objetos que no son útiles y a la destrucción del objeto NSAutoreleasePool le falta mucho tiempo, puede que la aplicación llegue a tener problemas de espacio en la memoria. En la imagen 3.12 se puede ver un ejemplo de la utilización del NSAutoreleasePool y la llamada autorelease.

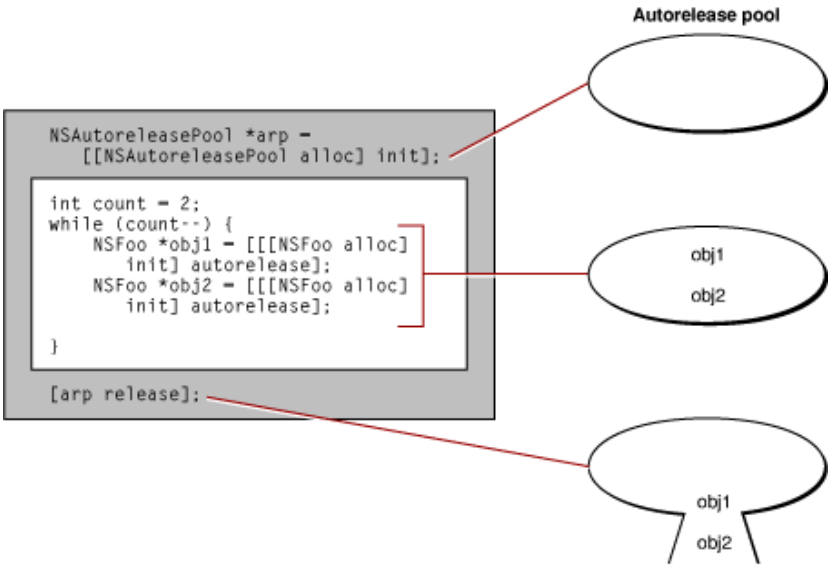

Imagen 3.12

3.6. Control táctil

Tal como se explicó en el inicio del apartado ciclo de vida de una aplicación, en un punto del inicio de una aplicación, el iOS inicia un sistema que consiste en un bucle que captura los eventos que se van produciendo, por ejemplo al tocar la pantalla, y los envía a diferentes objetos de la aplicación. En la imagen 3.13 se puede ver un ejemplo del proceo del evento al tocar la pantalla. Cuando se produce un evento este es encolado en una cola de eventos, estos eventos son procesados secuencialmente y enviados al bucle principal de eventos de la aplicación. Después el bucle envía el evento al objeto principal de la aplicación y este se encarga de gestionarlo entre los diferentes objetos de la aplicación.

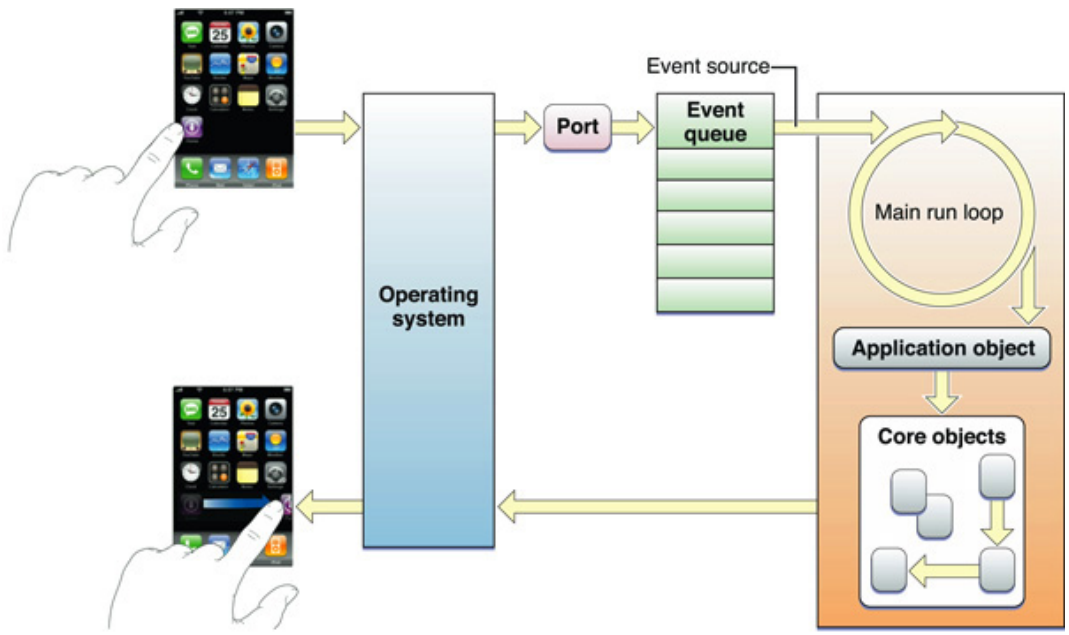

Imagen 3.12

Los eventos que son enviados a la aplicación son encapsulados en un objeto de la clase UIEvent que contiene información referente al evento. En el caso de los eventos que se producen al tocar la pantalla, el objeto del evento contiene un grupo de uno o más toques que se han producido y que están representados en objetos de la clase UITouch.

Los objetos que tienen la responsabilidad de gestionar y distribuir los eventos son aquellos que heredan de la clase UIResponder y concretamente son UIApplication, UIViewController, UIWindow y UIView. El primer objeto que recibe siempre el evento es la ventana principal de la aplicación (UIWindow) y este objeto se encarga de entregar el evento al objeto de primera respuesta. Normalmente el objeto de primera respuesta corresponde a uno de tipo vista (UIView) que normalmente es la vista donde tuvo lugar el toque de pantalla, aunque hay veces que puede ser un control (UIControl), como por ejemplo un botón, cuyo caso éste hará de objeto de primera respuesta.

En el caso de que el objeto de primera respuesta no responda al evento, el evento será entregado al siguiente objeto de respuesta. El siguiente objeto de respuesta siempre será la vista padre que contenía el objeto de primera respuesta. Si el siguiente objeto de respuesta tampoco responde, este continuará con la cadena de respuesta con el siguiente objeto padre que puede ser, por ejemplo, un controlador de vista (UIViewController). La cadena de respuesta acabará siempre con el objeto de aplicación principal (UIApplication).

4. COMUNICACIÓN

Anteriormente, en los capítulos 2 y 3, se han presentado y analizado el dispositivo y las herramientas necesarias para el proyecto, y después se ha realizado una introducción de los conceptos clave del desarrollo en dicho dispositivo. Teniendo esta base de conocimientos ya es posible abordar el desarrollo de las dos partes del proyecto final de carrera.

Este capítulo se centrará en el desarrollo de la primera parte del proyecto que corresponde al primer objetivo descrito en el capítulo 1. Haciendo memoria, este objetivo consiste en diseñar y desarrollar una librería estática que trabaje sobre el framework Game Kit —librería que contiene el código que realiza la conexión con otro iPhone— para que permita crear una red de iPhones. El capítulo se centrará en describir el funcionamiento y las limitaciones del Game Kit, escoger una metodología de desarrollo del software y seguir todos los procesos de esa metodología para obtener la librería.

4.1. Introducción

La comunicación inalámbrica entre dispositivos digitales no es una característica muy reciente, pero ha llegado hasta tal punto en que hoy en día se considera una característica fundamental. Los teléfonos móviles, los smartphones, las computadoras portátiles o de sobremesa, los routers, las impresoras, los coches, los equipos de música, los televisores, … y se podría continuar la lista de dispositivos que se pueden comunicar inalámbricamente entre si.

Un ejemplo muy familiar para cualquiera poseedor de un teléfono móvil es la de compartir una foto, un video o cualquier otro contenido con otros móviles mediante Bluetooth.

Siguiendo esta misma idea, la última generación de smartphones no solo han ofrecido este servicio, sino que han dado un paso adelante ofreciendo a los desarrolladores —en sus respectivas plataformas— la posibilidad de desarrollar aplicaciones con la opción de conexión inalámbrica a otros dispositivos. Normalmente, todas las funcionales que controlan la comunicación inalámbrica están encapsuladas en un framework que forma parte de sus respectivas SDK.

En el caso de Apple, en la liberación de la versión 3 del iOS trajo como novedad el framework denominado Game Kit. Este framework ofrece al desarrollador la capacidad de conectar y transmitir datos (y voz) con otros dispositivos con iOS mediante una conexión peer-to-peer a través de Bluetooth o de una red local Wi-Fi.

Más tarde, en la versión 4.1 del iOS, este framework aglutinó una nueva función, denominada Game Center. Esta nueva funcionalidad consiste en una red social en que se puede añadir una aplicación o juego desarrollado permitiendo al usuario del dispositivo añadir a otros usuarios, compartir mensajes y puntuaciones de los juegos, etc.

De aquí en adelante, en este capítulo se referirá al Game Kit simplemente por su capacidad de conectar con otros dispositivos y transmitir datos, y no por su capacidad social que apareció en la versión 4.1.

4.2. Framework Game Kit

El framework Game Kit[11] tiene la capacidad de realizar conexiones peer-to-peer que permiten crear una red ad-hoc Bluetooth o una red local inalámbrica entre varios dispositivos con iOS. Aunque en principio este diseñado para juegos, esta red permite intercambiar cualquier tipo de datos entre los diferentes usuarios de una misma copia de la aplicación.

Los requisitos que se han de cumplir para poder utilizar este framework son que el iOS sea la versión 3 o superior y que el dispositivo tenga Bluetooth —caso que no cumple la primer versión del iPhone y del iPod touch—.

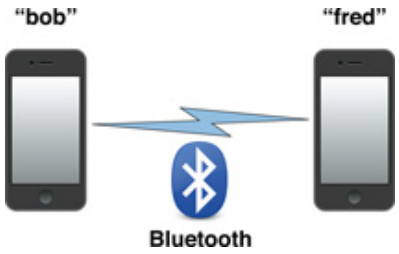

Imagen 4.1

La principal clase del framework se llama `GKSession`. Gracias a los objetos de esta clase se puede gestionar todo el ciclo de comunicación (búsqueda de dispositivos, conexión, envío y recepción de datos, y desconexión). Aunque la clase controle el sistema de recepción y envío de datos, no impone ninguna directriz en cuanto a cual debe ser el formato de los datos que se transmiten, dejando total libertad al desarrollador en diseñar el formato.

En el caso que el desarrollador no quiera implementar su propia interfaz de usuario para descubrir otras sesiones (otros dispositivos), puede utilizar la clase `GKPeerPickerController` para presentar una interfaz estándar que configurará una sesión.

4.2.1. Sesiones

Tal como se comentó en líneas más arriba, el objeto sesión se encarga de descubrir otras sesiones y de conectarse a ellas para formar una red. Una vez conectados en red, la sesión permite enviar datos a otros dispositivos. Además, el desarrollador deberá implementar un delegado para recoger las peticiones de conexión y para recibir los datos enviados por los otros dispositivos.

[11]http://developer.apple.com/library/ios/#DOCUMENTATION/NetworkingInternet/Conceptual/GameKit_Guide/GameKitConcepts/GameKitConcepts.html#//apple_ref/doc/uid/TP40008304-CH100-SW1

Peers

Los dispositivos iOS que estan conectados en una red inalámbrica se denominan “peers”. También se puede referir como “peer” al objeto sesión que se utiliza para realizar la comunicación. El objeto de la clase `GKSession` de cada peer de la red crea un identificador único (peer ID), de tipo string, que le identifica enfrente a los demás peers conectados. La interacción con los demás peers de la red se realiza con estos identificadores. Por ejemplo, si un peer quiere saber el nombre legible de otro peer puede llamar al método `displayNameForPeer:` donde debe introducir como parámetro de entrada el string del peer ID de ese peer.

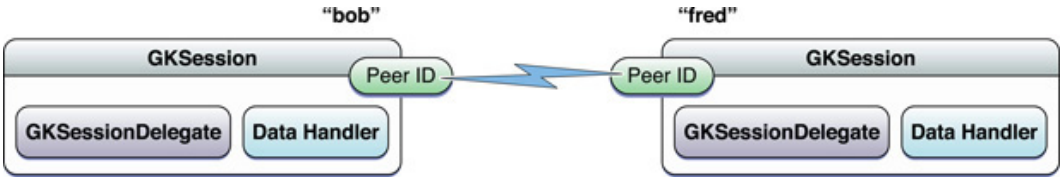

Imagen 4.2

Los demás peers de una red pueden variar su estado relativo de sus sesiones locales. Un peer puede aparecer o desaparecer de la red, conectarse o desconectarse de una sesión. El desarrollador puede implementar el método delegado `sesión:peer:didChangeState:` para capturar las notificaciones del cambio del estado de los peers de la red.

Descubrir otros peers

Dependiendo del tipo de aplicación que se desarrolle, el desarrollador tendrá la responsabilidad de determinar el tipo de configuración más adecuada para la sesión. La sesión descubre los otros peers de la red según la modalidad configurada en el momento de su inicialización. Existen tres modalidades:

- Servidor: la sesión actúa como servidor, anunciando que ofrece un tipo de servicio y esperando alguna conexión.
- Cliente: la sesión actúa como cliente, buscando algún anuncio de servicio de un servidor para poder conectarse.
- Peer: la sesión actúa tanto como servidor y cliente a la vez.

Los servidores anuncian sus servicios utilizando un identificador de sesión (session ID) de tipo string. Los clientes solo pueden buscar servidores que tengan el mismo identificador de sesión que el suyo.

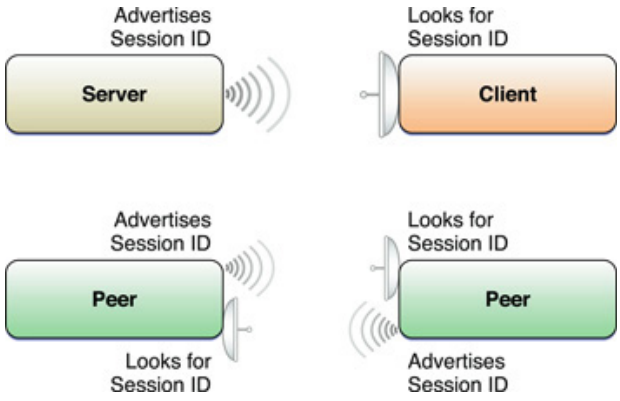

Imagen 4.3

Para establecer una conexión como mínimo es necesario que uno de los dispositivos sea servidor para que el otro busque su servicio.

Implementar un servidor

Una instancia de la aplicación actúa como servidor inicializando el objeto sesión con la llamada `initWithSessionID:displayName:sessionMode:` especificando el parámetro de modalidad de sesión (sessionMode) con las constantes `GKSessionModeServer` o `GKSessionModePeer`. Después, para que el servidor anuncie su servicio, debe configurar el atributo `available`, del objeto sesión, a `YES`.

Los servidores han de notificar cuando reciben una petición de conexión de un cliente. Cuando un cliente envía una petición de conexión, el método delegado `session:didReceiveConnectionRequestFromPeer:` del servidor es llamado. El comportamiento normal del delegado es usar el `peerID` (el segundo parámetro de entrada) para realizar la llamada `displayNameForPeer:` para saber el nombre del cliente y, de esta manera, notificarlo por la interfaz de usuario para que el usuario decida si aceptar o no la conexión.

El delegado podrá aceptar la petición de conexión llamando al método `acceptConnectionFromPeer:error:` o denegarlo con el método `denyConnectionFromPeer:` del objeto `GKSession`.

Conectarse a un servidor

Una instancia de la aplicación actúa como cliente inicializando la sesión con la llamada `initWithSessionID:displayName:sessionMode:` especificando el parámetro de modalidad de sesión (sessionMode) con las constantes `GKSessionModeClient` o `GKSessionModePeer`. Después, para que el cliente busque al servicio de un servidor, debe configurar el atributo `available`, del objeto sesión, a `YES`.

Cuando un cliente descubre un servidor disponible, la sesión cliente recibe la llamada del método delegado `session:peer:didChangeState:`. Con el string `peerID` (el segundo parámetro de entrada del método anterior) del servidor descubierto se podrá llamar al método

`displayNameForPeer:` para saber el nombre del servidor y notificarlo por la interfaz de usuario. Cuando el usuario seleccione al servidor (o peer) que se quiere conectar, la aplicación llamará al método de sesión `connectToPeer:withTimeout:` para realizar una petición de conexión.

Cuando la conexión sea creada satisfactoriamente, el método delegado `session:peer:didChangeState:` será llamado para informar que el nuevo peer está conectado.

Intercambiar datos

Los peers conectados a una sesión pueden intercambiar datos con otros peers conectados. La aplicación podrá enviar datos a todos los peers conectados mediante el método `sendDataToAllPeers:withDataMode:error:` o a un subconjunto de peers mediante el método `sendData:toPeers:withDataMode:error:`. Los datos enviados a otros peers son capsulados en objetos del tipo `NSData`. Se puede diseñar y usar cualquier tipo de formato de datos, incluso crear tus propios formatos. A pesar de que se pueden enviar paquetes de datos de hasta 87 kilobytes, por razones de rendimiento se recomienda solo enviar hasta 1.000 bytes. Para aquellos paquetes mayores de 1.000 bytes se recomienda particionar el paquete de datos en pequeños paquetes y unirlos en el dispositivo destinatario.

Se puede escoger entre dos modos de envío de datos (especificado en el parámetro `withDataMode:`): la forma fiable (reliably), cuyo caso la sesión se encargaría automáticamente retransmitir aquellos datos que no hayan podido llegar a su destino, o la forma poco fiable (unreliably), cuyo caso la sesión solo enviaría los datos una vez. Los envíos de la forma poco fiable son utilizados para datos que deben ser recibidos en tiempo real, siendo más importante enviar paquetes con nuevos datos que recibir paquetes reenviados por un problema de envío. En cambio, los envíos de forma fiable son utilizados para asegurar que los datos enviados llegarán a su destino en el mismo orden en que fueron enviados.

A medida que los datos son enviados, los peers destinatarios recibirán la llamada del método delegado `receiveData:fromPeer:inSession:context:`. El desarrollador deberá implementar este método para gestionar los datos recibidos y así poderlos usar en su aplicación. Un requisito previo para poder recibir paquetes de datos es llamar al método `setDataReceiveHandler:withContext:` para indicar cual es el objeto en que se haya implementado el método de recepción.

Un apunte muy importante es que este framework no tiene implementado ningún aspecto de seguridad. Por lo tanto, la recepción de los datos enviados por otros peers se deben considerar como datos inseguros. El desarrollador debe validar los datos recibidos por otros peers.

Desconectarse de un peer

Para finalizar una sesión y desconectarse de todos los peers, se debe llamar al método `disconnectFromAllPeers`. En cambio, si se quiere desconectar de un solo peer y mantener la conexión con otros peers conectados se debe usar el método `disconnectPeerFromAllPeers:`.

El comportamiento natural de estas redes son poco fiables, en consecuencia, si un peer no responde durante un periodo de tiempo este es desconectado automáticamente de la red. El tiempo de espera antes de la desconexión se puede modificar en el atributo `disconnectTimeout`.

Limpieza

Cuando la sesión esté lista para eliminarla, el desarrollador debe desconectarse de todos los peers, establecer el valor del atributo `available` a NO, quitar la referencia del objeto delegado receptor de datos y del objeto delegado normal, y liberar el objeto de sesión.

4.2.2. Ejemplo de código de sesiones

En este subapartado se muestran trozos de código como ejemplo de las características descritas en el subapartado anterior.

1 – Creación y inicialización de la sesión.

```
GKSession* session = [[GKSession alloc] initWithSessionID:myExampleSessionID
displayName:myName sessionMode:GKSessionModePeer];

// se asume que este objeto será el objeto delegado de sesión y de recepción
// de datos.
session.delegate = self;
[session setDataReceiveHandler: self withContext:nil];
```

2 – Implementación del método delegado `session:peer:didChangeState:` de sesión.

```
- (void)session:(GKSession *)session peer:(NSString *)peerID
didChangeState:(GKPeerConnectionState)state
{
	switch (state)
	{
		case GKPeerStateConnected:
		// Registrar el peerID del otro peer.
		// Informar a la aplicación que un peer se ha conectado.
			break;
		case GKPeerStateDisconnected:
		// Informar a la aplicación que un peer se ha desconectado.
			break;
	}
}
```

3 – Enviar datos a otros peers.

```
- (void) mySendDataToPeers: (NSData *) data
{
	[session sendDataToAllPeers: data withDataMode: GKSendDataReliable
	error: nil];
}
```

4 – Recibir datos de otros peers.

```
- (void) receiveData:(NSData *)data fromPeer:(NSString *)peer inSession:
(GKSession *)session context:(void *)context
{
	// Leer los bytes de data y realizar alguna acción especifica.
}
```

5 – Cuando es el momento de poner fin la conexión, se desconecta de los demás peers y se libera la sesión.

```
[session disconnectFromAllPeers];
session.available = NO;
[session setDataReceiveHandler: nil withContext: nil];
session.delegate = nil;
[session release];
```

4.2.3. Pruebas

En la documentación oficial de Apple no aparece ninguna indicación de cómo se deben hacer las pruebas de comunicación de este framework.

En un principio, la forma más correcta de realizar las pruebas es conectar varios dispositivos (reales) de desarrollo consiguiendo simular una situación la más real posible. El principal problema de esta forma es la disponibilidad de más de un dispositivo para poder hacer las pruebas, no todos los desarrolladores tienen disponible más de un dispositivo.

Existe otra alternativa a las pruebas con dispositivos reales y es la utilización de varios simuladores (la aplicación procedente del SDK que simula un dispositivo iOS) conectados en una misma red local Wi-fi y/o ethernet. Por suerte, el framework no solo funciona con conexiones por Bluetooth, sino que también se puede utilizar con conexiones Wi-fi (tanto el simulador como el dispositivo real), aunque con el inconveniente que los dispositivos iOS no pueden crear redes Wi-fi, solo pueden unirse a una. Siguiendo este sistema podemos, por ejemplo, conectar en una misma red un dispositivo real (por Wi-fi) y dos simuladores (por ethernet) para realizar las mismas pruebas que haríamos con tres dispositivos reales. A pesar de todo, esta alternativa también tiene varios inconvenientes:

- La necesidad de utilizar una máquina con Mac OS X por cada simulador que se utilice. Este inconveniente surge debido a que la aplicación solo deja ejecutar una instancia del simulador, con la consecuencia que si se necesita varios simuladores se deberá usar varias máquinas con Mac OS X.
- El rendimiento de las pruebas sobre una red Wi-fi y/o ethernet es diferente respecto a una red Bluetooth utilizado en los dispositivos reales. Por ejemplo, una de las diferencias de rendimiento entre las dos redes son los tiempos de espera de conexión o de búsqueda de dispositivos.
- Dependencia directa del funcionamiento del “router” que gestiona la red. Normalmente este punto no es un inconveniente porque es raro encontrarse con un “router” que tenga problemas de funcionamiento, pero no es imposible. Los problemas que puede causar un “router” con problemas de funcionamiento pueden ser variados, desde problemas de descubrimiento de nuevos peers disponibles, hasta problemas de transmisión de paquetes de información que no llegan a su destino.

4.3. Limitaciones del Game Kit y del Bluetooth

Una vez acabada la lectura de la documentación oficial del Game Kit se pueden observar una serie de aspectos que han sido poco especificados o parcialmente ausentes y que, en su posterior búsqueda y análisis de estos aspectos, nos revelan los límites de este framework.

El primer aspecto que no se especifica claramente en la documentación es que **tipos de arquitecturas de red** se pueden utilizar en el framework. Consciente de ello, Apple añadió un aviso explicando que en su documentación no cubre detalles que impliquen aspectos del diseño de las aplicaciones o juegos en red y que solamente documentan el funcionamiento de cada clase del framework.

A pesar de este pequeño inconveniente, en la descripción del funcionamiento de una sesión de la documentación del Game Kit, en el cual explica que hay tres modalidades de configuración de una sesión (vea apartado *4.2.1.* de este documento), indican indirectamente dos arquitecturas de red que se pueden utilizar. La primera arquitectura es la de cliente-servidor, en la cual hay un dispositivo que hace el papel de servidor y varios haciendo el papel de clientes, donde tenemos una conexión de tipo punto-a-multipunto; y la segunda arquitectura es la peer-to-peer simple, en cual hay solo dos dispositivos conectados entre si, los cuales ambos hacen los papeles de cliente y servidor y donde tenemos una conexión de tipo punto-a-punto. En la actualidad, todas las aplicaciones del iOS que tienen funciones de conectividad entre dispositivos tienen implementado una de estas dos arquitecturas de red.

Imagen 4.4

El inconveniente que tiene la arquitectura cliente-servidor a la hora de realizar redes con más de dos dispositivos es la dependencia que tienen los clientes sobre el servidor en la estructura de la red. Esta dependencia conlleva que en el caso que el servidor desaparezca, también desaparecería toda la red y, por lo tanto, la necesidad de crear toda una nueva red para volver a conectar todos los demás dispositivos.

El segundo aspecto que tampoco especifica claramente es el **número de conexiones** que puede mantener un dispositivo. Originalmente en la documentación de Game Kit no especificaba nada, pero en la ultima revisión del documento, realizada en el 2011, Apple añadió una nota en que especificaba que el número máximo de clientes-servidor es de 16 dispositivos. Por desgracia, esta nota es muy ambigua porque no especifica en ningún momento para que tipo de conexión (Bluetooth o Wi-fi) es este límite de dispositivos.

Ante la falta de respuestas se realizó una pequeña búsqueda por internet. En primer lugar se comprobó en la web del Bluetooth SIG[12] (asociación de empresas que controla las especificaciones del Bluetooth) la documentación con las especificaciones del Bluetooth v2.0[13]. Este documento especifica que un dispositivo puede conectarse como máximo con siete dispositivos a la vez, en una conexión punto-a-punto.

En segundo lugar, se investigó por la web oficial de Apple páginas que contuvieran el término Bluetooth. Entre las páginas encontradas llamó la atención una titulada *Dispositivos de entrada inalámbricos: preguntas frecuentes sobre Bluetooth*[14], ya que contenía la respuesta a nuestra pregunta. La respuesta ofrecida era la siguiente: *Según las especificaciones Bluetooth oficiales, el número máximo de dispositivos Bluetooth que pueden conectarse a la vez es siete. Sin embargo, tres o cuatro dispositivos es un límite práctico, en función del tipo de dispositivos y los perfiles que se utilicen.* A pesar que esta web está dirigida a ordenadores Mac, la información obtenida nos sirve porque coincide con las especificaciones de Bluetooth vistas anteriormente.

Por lo tanto, se puede afirmar que el límite de dispositivos conectados punto-a-punto a un dispositivo mediante **Bluetooth** son **entre 3 y 4** dispositivos. A partir de este dato se puede imaginar que el máximo de 16 dispositivos que se indicaba en la documentación de Game Kit hacia referencia a conexiones mediante Wi-fi.

A pesar del gran número de dispositivos que se pueden conectar mediante Wi-fi, se ha de mencionar que existe el inconveniente de que todos los dispositivos necesitan estar conectados en una red Wi-fi creada mediante un router inalámbrico, ya que los dispositivos iOS no tienen la capacidad de crear redes Wi-fi (solo tiene la capacidad de unirse).

Recopilando toda la información anterior, se puede resumir que el framework Game Kit sólo puede conectar hasta cuatro dispositivos cuando está utilizando el Bluetooth y la arquitectura cliente-servidor y que en el caso que desapareciera el dispositivo que hace de servidor toda la red también desaparecería.

4.4. Metodología de desarrollo

4.4.1. Metodología de desarrollo en cascada

Se trata de una metodología de trabajo compuesta por una serie de etapas lineales muy rígidas. No se puede iniciar una etapa del ciclo de vida de la aplicación hasta que la anterior se haya finalizado. Un caso típico es aquel en el que tenemos las siguientes etapas:

[12] http://www.bluetooth.org
[13] Core Version 2.0 + EDR - https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=40560
[14] http://support.apple.com/kb/HT3887?viewlocale=es_ES&locale=es_ES#howmany

- Análisis de requerimientos: Se analizan las necesidades de los usuarios finales del software para determinar qué objetivos debe cubrir. De esta fase surge una memoria, la cual contiene la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos.
- Especificación: Como resultado surge un documento que contiene la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras.
- Diseño: Se descompone y organiza el sistema en elementos que puedan elaborarse por separado. Es la fase donde se realizan los algoritmos necesarios para el cumplimiento de los requerimientos del usuario así como también los análisis necesarios para saber que herramientas usar en la etapa de Codificación.
- Codificación: Es la fase de programación o implementación propiamente dicha. Aquí se implementa el código fuente, haciendo uso de prototipos así como pruebas y ensayos para corregir errores.
- Pruebas: Se verifica la conformidad del software codificado con respecto a los requisitos analizados y consensuados en la memoria pactada en la fase de análisis de requisitos. Existen diferentes tipos de pruebas: Funcionales, unitarias, de stress, etc.
- Mantenimiento: Última fase del ciclo de vida del proyecto, el en que se resuelven incidencias.

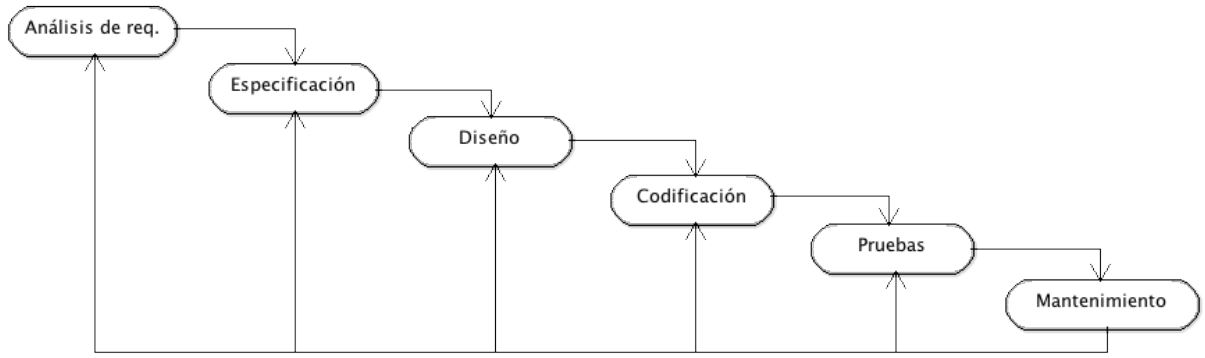

Imagen 4.5

Las ventajas que comporta seguir esta metodología de trabajo son: fácil seguimiento del proyecto debido a que es sencillo organizarlo, ya que sus fases no se mezclan; y la menor dificultad de inicio del proyecto si se conocen bien todos los requisitos. La principal desventaja de esta metodología es la alta rigidez de las etapas del proceso, haciendo que la detección de errores de especificación y/o diseño no sean detectados hasta llegar la fase de pruebas. Por esta razón, la mayoría de veces no se sigue completamente este proceso lineal.

4.4.2. Decisión

Para la ejecución de la primera parte del proyecto, correspondiente al primer objetivo, se ha decidido seguir la metodología de desarrollo en cascada o también conocido como ciclo de vida clásico. La principal razón en que se ha escogido esta metodología es porque se conoce de

antemano los requisitos del sistema y de la especificación, ya que son más o menos los mismo que ofrece el framework Game Kit.

4.5. Especificación

4.5.1. Definición del sistema

Visto los límites del Game Kit que se han descrito en apartados anteriores, se ha decidido desarrollar una librería de comunicación que supere estos límites y que, al igual que el oficial, sea fácil de utilizar en una aplicación que requiera la creación una red local con Bluetooth. Para realizar este propósito, la librería a desarrollar tomará como base o núcleo la librería de Game Kit.

Las características que se desea que tenga la red creada por la librería a desarrollar son las siguientes:

- El número de dispositivos conectados en la red debe ser mayor de 4 dispositivos.
- Tener la capacidad de recuperación o restablecimiento de la red después de la desconexión de cualquier de los dispositivos que la forman. Si el dispositivo desconectado quiere volver a unirse a la red, se deberá facilitar un mecanismo para que continúe la comunicación tal como estaba antes. La tarea de recuperación o restablecimiento de la red debe implicar el menor número posible de dispositivos.
- Encapsular el código en forma de librería para que sea usada de la misma manera que la librería Game Kit.

Además de estas características, la nueva librería debe ofrecer casi las mismas funcionalidades básicas que también ofrece la librería Game Kit. Estas funcionalidades básicas son las siguientes:

- Crear o inicializar una nueva red local.
- Buscar y unirse a una red ya creada.
- Desconectarse de una red que se esté conectado.
- Enviar y/o recibir datos de una red que se esté conectado.
- Recibir cambios de estado de una red que se esté conectado. Estos cambios serán simplemente conexiones y desconexiones de dispositivos.
- Información de los dispositivos conectados. La información ofrecida será un número identificador y un nombre de los dispositivos conectados.

4.5.2. Análisis de requisitos

La etapa de análisis de requisitos es aquella que define las necesidades de un usuario u otro sistema (en este caso, una aplicación) para solucionar un problema o para completar un objetivo.

Estas necesidades o requisitos, cuando estamos definiendo un sistema software, pueden ser de dos tipos: funcionales y no funcionales.

Los requisitos funcionales son aquellos procesos y datos que el sistema necesita para que en el momento de recibir una entrada pueda generar una salida. Para definir cuales son los requisitos funcionales de un sistema es habitual utilizar los casos de uso, ya que estos dos están directamente relacionados. Mediante esta relación, se puede sacar de cada casos de uso cual es el suceso y su funcionalidad que, en serie o en cadena, satisfacen las necesidades (o requisitos) del usuario.

Los requisitos no funcionales son una serie de cualidades o propiedades generales que debe tener el sistema para realizar su función. Estas cualidades pueden ser utilizadas por los usuarios para definir el nivel de satisfacción. Al igual que los requisitos funcionales pero con una relación indirecta, alguno de estos requisitos pueden estar relacionado con algún caso de uso.

En cada requisito analizaremos los siguientes parámetros:

- Requisito : identificador del requisito. RF, requisito funcional. RNF, requisito no funcional.
- Suceso / caso de uso : caso de uso en el cual se relaciona el requisito.
- Descripción : descripción del requisito.
- Justificación : motivo para cumplir este requisito.
- Dependencia : el requisito puede depender o requerir de otros requisitos del sistema.

Requisitos funcionales

Requisito: RF1	**Suceso / caso de uso:** Crear una red, formar parte de una red.
Descripción: El sistema debe iniciar una nueva red desde cero.	
Justificación: El sistema debe permitir al usuario (o aplicación) obtener una red donde inicialmente solo formará parte este dispositivo.	
Dependencia: ---	

Requisito: RF2	**Suceso / caso de uso:** Buscar una red, unirse a una red, formar parte de una red.
Descripción: El sistema debe buscar una red abierta y posteriormente unirse a ella.	
Justificación: El sistema mostrará al usuario (o aplicación) una lista de las redes disponibles para que posteriormente el usuario pueda seleccionar una e iniciar la conexión (unirse). Una vez unido el dispositivo será un miembro más de la red seleccionada.	
Dependencia: ---	

Requisito: RF3	**Suceso / caso de uso:** Desconectarse de una red.
Descripción: El sistema debe realizar la desconexión de una red en que forme parte.	
Justificación: El sistema notificará a cada dispositivo miembro de la red que se va desconectar de la red de que forma parte. Una vez realizada la desconexión, el sistema informará al usuario (o aplicación) que este dispositivo ya no pertenece a dicha red.	
Dependencia: RF1, RF2	

Requisito: RF4	**Suceso / caso de uso:** Enviar datos.
Descripción: El sistema debe enviar una serie de datos en una red en que forme parte.	
Justificación: El sistema debe enviar una serie de paquetes de datos seleccionado por el usuario (o aplicación) a uno o más dispositivos de que formen parte de la misma red que él.	
Dependencia: RF1, RF2	

Requisito: RF5	**Suceso / caso de uso:** Recibir datos.
Descripción: El sistema debe recibir una serie de datos de una red en que forme parte.	
Justificación: El sistema debe notificar al usuario (o aplicación) que ha recibido una serie de paquetes de datos enviados por uno de los dispositivos de que forme parte de la misma red que él.	
Dependencia: RF1, RF2	

Requisito: RF6	**Suceso / caso de uso:** Recibir cambios de estado de una red.
Descripción: El sistema debe recibir cualquier aviso de cambio del estado de una en que forme parte.	
Justificación: El sistema debe notificar al usuario (o aplicación) de cualquier conexión o desconexión de un dispositivo que se produzca en la misma red que él sea miembro.	
Dependencia: RF1, RF2	

Requisito: RF7	**Suceso / caso de uso:** Recibir información de los dispositivos conectado a una red.
Descripción: El sistema debe recopilar datos de los miembros de una red en que forme parte.	
Justificación: El sistema notificará al usuario (o aplicación) una lista de los identificadores y nombres de los dispositivos en que formen parte de la misma red que él.	
Dependencia: RF1, RF2	

Requisitos no funcionales

Unos párrafos más arriba se había descrito que los requisitos no funcionales podrían ser también obtenidos, indirectamente, de los casos de uso, pero en este caso no ha sido así. En este caso, los requisitos no funcionales han sido sacados directamente del apartado *definición del sistema* y son aplicados sobre todo el sistema en general.

Reusabilidad

Requisito: RNF1	Suceso / caso de uso: -
Descripción: Capacidad del producto software para funcionar como bloques básicos de la construcción de diferentes aplicaciones.	
Justificación: Se desea que todo el sistema sea encapsulado de forma que pueda ser utilizado en diferentes aplicaciones que necesiten características de comunicación entre dispositivos.	
Dependencia: -	

Usabilidad

Requisito: RNF2	Suceso / caso de uso: -
Descripción: Conjunto de atributos relacionados con el esfuerzo necesario para su uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios.	
Justificación: Se desea que el sistema sea utilizado de forma similar a la librería Game Kit para que los desarrolladores no necesiten grandes esfuerzos para su uso.	
Dependencia: -	

Recuperabilidad (Fiabilidad)

Requisito: RNF3	Suceso / caso de uso: -
Descripción: Capacidad del producto software para restablecer un nivel de prestaciones-especificado y de recuperar los datos directamente afectados en caso de fallo.	
Justificación: Se desea que el sistema restablezca (recupere) la capacidad de comunicación de la red después de ser afectado por la desconexión de cualquier de los dispositivos que la forman.	
Dependencia: -	

4.5.3. Especificación – casos de uso

Los casos de uso son un conjunto de secuencias de acciones y interacciones relacionadas que describen de que manera los actores (usuarios o otros sistemas) usan el sistema per completar un determinado objetivo. Tal como se describió en el apartado anterior, los casos de uso están relacionados con los requisitos funcionales.

Diagrama de casos de uso

En el siguiente diagrama de casos de uso (Imagen 4.6) se describe la iteración de los nueve casos que forma el sistema descrito anteriormente con dos actores.

Los dos actores que intervienen son: **la aplicación** que usa el sistema para obtener capacidad de comunicación con otros dispositivos que tenga la misma aplicación, y **la librería de comunicación nativa** (Game Kit) que se encarga de realizar las operaciones básica de comunicación que el sistema necesita.

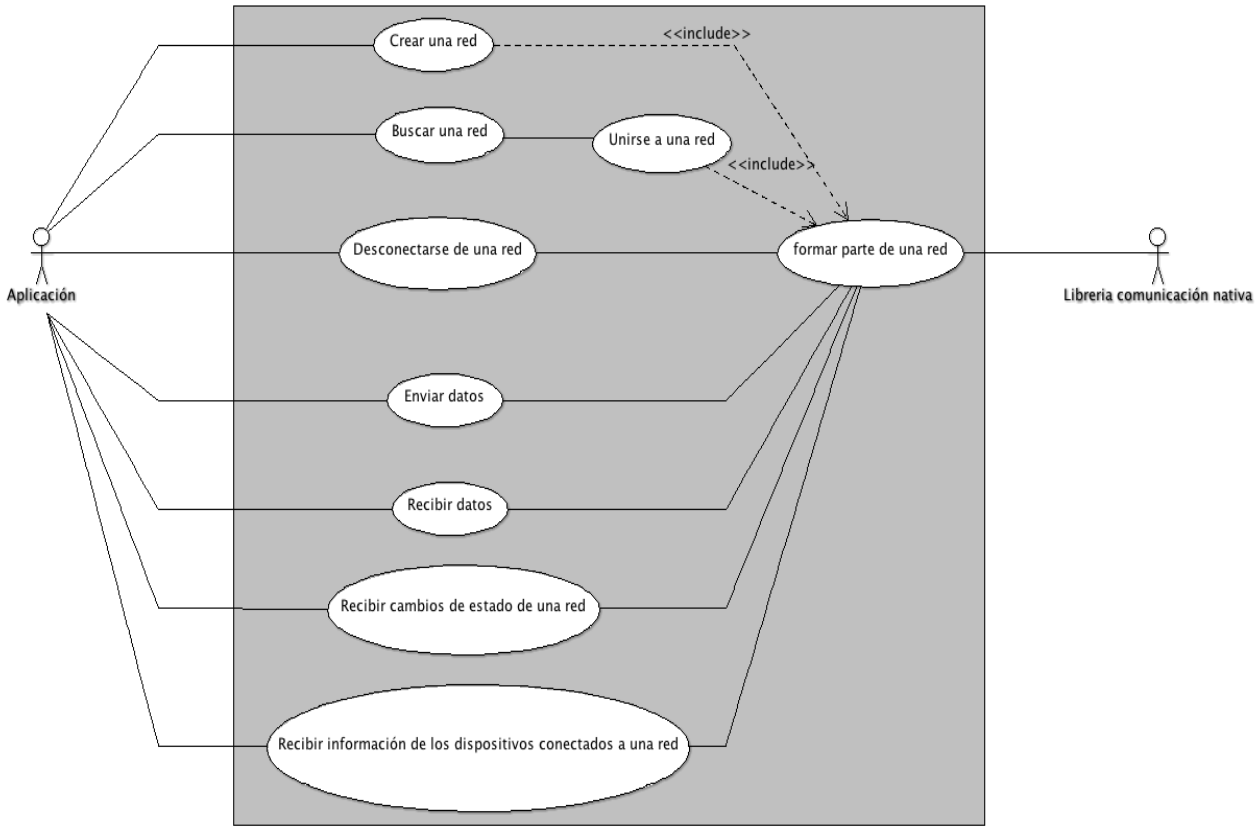

Imagen 4.6

Descripción de los casos de uso

A continuación se mostrará la descripción de cada uno de los caso de uso del diagrama mostrado en el apartado anterior (Imagen 4.6).

Caso de uso: Crear una red
Descripción: Se crea una nueva red desde cero que inicialmente solo tiene un dispositivo que forme parte.
Actores: Aplicación
Precondiciones: -
Inclusión de otros casos de uso: "formar parte de una red"
Dialogo típico: La aplicación llamará al método de creación de una red, rellenando una serie de parámetros como por ejemplo el nombre en el cual quiere ser identificado dentro de la red que va a crear.

Caso de uso: Buscar una red
Descripción: Se realiza una búsqueda de aquellas redes que están disponibles y abiertas para unirse a ella.
Actores: Aplicación
Precondiciones: -
Inclusión de otros casos de uso: -
Dialogo típico: La aplicación llamará al método de búsqueda de redes para obtener una lista de aquellas que están disponibles y abiertas a nuevas incorporaciones. La aplicación elegirá una de las redes, si ha encontrado alguna, para que posteriormente se una a ella.

Caso de uso: Unirse a una red
Descripción: Se realiza la petición de unión de la red en cuestión para que posteriormente se forme parte de ella.
Actores: -
Precondiciones: Haber encontrado una red en el caso de uso "buscar una red".
Inclusión de otros casos de uso: "formar parte de una red"
Dialogo típico: Después que la aplicación haya seleccionado una red en el caso de uso "buscar una red", llamará al método que realiza la petición de unión a la red. En el caso que se permita la unión, el dispositivo formará parte de la red en cuestión.

Caso de uso: Formar parte de una red
Descripción: se confirma que ya se forma parte de una red.
Actores: Librería de comunicación nativa
Precondiciones: Solo puede ser incluido en los casos de uso que necesiten obtener acceso a una red, que son "crear una red" y "unirse a una red".
Inclusión de otros casos de uso: -
Dialogo típico: La librería de comunicación nativa da soporte a la formación de la red con los diferentes dispositivo que la componen y permite realizar las funcionalidades de los siguientes casos de uso. La incluso de este caso de uso en "crear una red", permite que otros dispositivos vean que esta red es disponible a nuevas incorporaciones de dispositivos.

Caso de uso: Desconectarse de una red
Descripción: Se realiza la desconexión de una red en que se forme parte.
Actores: Aplicación
Precondiciones: Formar parte de la red que se quiere desconectar. – "formar parte de una red"
Inclusión de otros casos de uso: -
Dialogo típico: La aplicación, que previamente creó o se unió a una red, llamará al método de desconexión para salir de esa red.

Caso de uso: Enviar datos
Descripción: Se envía una serie de datos a un dispositivo.
Actores: Aplicación
Precondiciones: Que el dispositivo emisor y receptor pertenezcan a la misma red. – "formar parte de una red"
Inclusión de otros casos de uso: -
Dialogo típico: La aplicación, que previamente creó o se unió a una red, llamará al método de envío para enviar una serie de datos a otro dispositivo que pertenezca a la misma red que él.

Caso de uso: Recibir datos
Descripción: Se recibe una serie de datos de un dispositivo.
Actores: Aplicación
Precondiciones: Que el dispositivo emisor y receptor pertenezcan a la misma red. – "formar parte de una red"
Inclusión de otros casos de uso: -
Dialogo típico: La aplicación, que previamente creó o se unió a una red, recibirá el evento de recepción para recibir una serie de datos enviados por un dispositivo que pertenece a la misma red que él.

Caso de uso: Recibir cambios de estado de una red
Descripción: Se recibe información relativa a cambios que se produce en una red.
Actores: Aplicación
Precondiciones: Formar parte de la red que se producen los cambios de estado. – "formar parte de una red"
Inclusión de otros casos de uso: -
Dialogo típico: La aplicación, que previamente creó o se unió a una red, recibirá el evento de que un dispositivo se ha unido o desconectado de la misma red que él.

Caso de uso: Recibir información de los dispositivos conectados a una red
Descripción: Se obtiene información relativa a los dispositivos conectados a una red.
Actores: Aplicación
Precondiciones: Formar parte de la red que se quiere obtener la información. – "formar parte de una red"
Inclusión de otros casos de uso: -
Dialogo típico: La aplicación, que previamente creó o se unió a una red, llamará al método para obtener el listado de dispositivos conectados en la red con sus datos. Estos datos serán por ejemplo, el nombre en que se dará a conocer el dispositivo y su identificación.

4.5.4. Especificación – diagrama conceptual de datos

El diagrama conceptual de datos sirve para representar los datos que manejará el sistema y cómo están relacionados entre ellos. Estos datos están representados en forma de objetos que tendrán una serie de atributos. La relación entres los datos o objetos estarán definidos por una serie de representaciones lineales y restricciones que globalmente representarán al sistema.

En el caso de este desarrollo, la variedad de datos que se maneja es tan pequeña que se podría haber omitido su representación. Aun así, se procederá a mostrarlos en el siguiente diagrama:

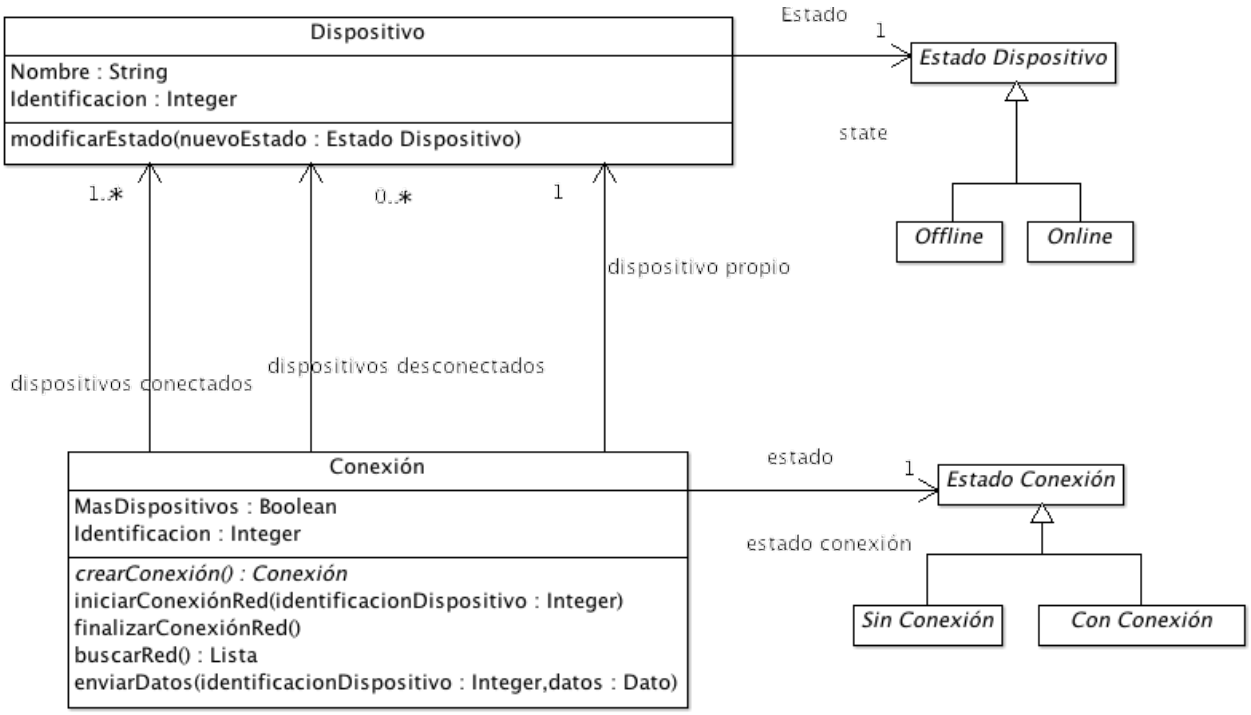

Imagen 4.7

Restricciones textuales:

- Llave externa: (Dispositivo: Identificación), (Conexión: Identificación)
- Una instancia de Dispositivo puede estar en la lista de “dispositivos conectados” o en la lista de “dispositivos desconectados”, no puede estar en las dos a la vez.
- Si una instancia de Dispositivo esta en estado Offline, este debe estar en la lista de “dispositivos desconectados”.
- Si una instancia de Dispositivo esta en estado Online, este debe estar en la lista de “dispositivos conectados”.
- Cuando se crea una instancia de Conexión, automáticamente se creará una instancia de Dispositivo que se relacionará como “dispositivo propio”.

4.6. Diseño

Una vez acabada la etapa de especificación, el desarrollo está en el punto en que se sabe cual es el sistema que se quiere construir y lo que debe hacer. Ahora, en la etapa de diseño, se debe plantear como lo debe hacer (o funcionar) el sistema. Para conseguir esta finalidad, se debe definir que arquitecturas, protocolos, patrones y clases se van a utilizar.

Normalmente, una de las primeras decisiones que se debe tomar es la elección de la tecnología que se va utilizar, pero en este caso, la decisión ya fue tomada en el momento en que se definió el proyecto final de carrera. En la definición del proyecto se estableció que se utilizaría el iPhone. Debido a esto, el desarrollo se hará bajo el sistema operativo iOS y, por lo tanto, el código del proyecto también podrá ser utilizado en el iPod Touch y en el iPad.

4.6.1. Arquitectura de red

En un punto del apartado 4.5.1. (*definición del sistema*), se describió cuales debían ser las características de la red creada por la librería a desarrollar. Estas características no se han podido reflejar en la etapa de especificación debido a que estas describen la forma que debería funcionar la red.

Para el diseño de la arquitectura de red, las características comentadas anteriormente son requisitos que deben cumplir en el diseño de la arquitectura. Los requisitos que se describieron en el apartado 4.5.1. son los siguientes:

- El número de dispositivos conectados en la red debe ser mayor de 4 dispositivos.
- Tener la capacidad de recuperación o restablecimiento de la red después de la desconexión de cualquier de los dispositivos que la forman. Si el dispositivo desconectado quiere volver a unirse a la red, se deberá facilitar un mecanismo para que continúe la comunicación tal como estaba antes. La tarea de recuperación o restablecimiento de la red debe implicar el menor número posible de dispositivos.

La arquitectura de servidor-cliente no cumple estos requisitos por las razones descritas en el apartado 4.3. . Por lo tanto, se debe buscar un nuevo diseño arquitectónico de red que cumpla estos requisitos.

La arquitectura de red que se ha escogido es la siguiente:

Imagen 4.8

La arquitectura mostrada en la imagen anterior se denomina arquitectura **en cadena** (también conocida como “en serie”). Las dos variantes de ésta sencilla arquitectura son la lineal y la anilla. La diferencia entre las dos variantes es que la lineal siempre tendrá dos nodos que harán de extremos, mientras que la variante de anilla no tiene extremos ya que todos los nodos están unidos entre si formando un circulo. De las dos variantes se ha escogido **la lineal** . A continuación se va enumerar las diferentes razones que han hecho escoger esta arquitectura:

- Cada dispositivo o nodo que forma la red tendrá como **máximo dos conexiones** (punto-a-punto) o vecinos. Al tener solo dos conexiones en cada dispositivo se consigue que ninguno de los nodos nunca pase del limite de cuatro conexiones punto-a-punto que se había descrito en el apartado 4.3..Por lo tanto, si se une el concepto de conexión en cadena con lo descrito

anteriormente se concluye que teóricamente esta arquitectura permitiría construir una red de iPhones sin limite de miembros.

- En el caso de una desconexión en la red y tal como se puede deducir del punto anterior , el número de dispositivos o nodos afectados seria como máximo dos (en ningún momento se cuenta como afectado el nodo que se ha desconectado). También se puede ver que a consecuencia de la desconexión, la red se divide en dos subredes. Para volver a unir las dos subredes simplemente se necesita realizar una reconexión de los dos nodos afectados. Por lo tanto, el número de acciones necesarias para **restablecer la red** ante una desconexión es solo de **uno**.
- La **incorporación de nuevos nodos** o dispositivos se realizan directamente en un de los dos **extremos** sin necesidad de otras operaciones. En cambio en la variante de anilla, es necesario realizar dos operaciones adicionales para la incorporación de nuevos nodos, la obertura y el cierre de la anilla.

Aparte de las características que ofrece esta arquitectura, se ha decido aplicar un criterio para las incorporaciones de nuevos nodos. Las incorporaciones de nuevos nodos siempre se realizarán en el mismo extremo de la red con el fin de establecer un sentido único en las conexiones. Siguiendo este criterio se obtiene dos resultados positivos: fortalecer la arquitectura al darle un orden y sentido de conexión que será muy útil para la recuperación de la red después de una desconexión; y que cada nodo incorporé un nuevo nodo solo una vez.

Imagen 4.9

En la imagen 4.9 se puede ver un ejemplo de una incorporación. El dispositivo del extremo derecho se ha conectado al dispositivo que era ultimo (o extremo) de la red, haciendo que el nuevo dispositivo sea ahora el dispositivo que debe permitir la conexión del próximo dispositivo. El dispositivo que permitió conectarse al nuevo se convierte en un nodo normal de la red que no aceptará ninguna nueva conexión.

4.6.2. Protocolos

Además de la definición de una arquitectura, es necesario definir una serie de protocolos de control que permitan el correcto funcionamiento de la red. Hay que recordar que muchos de los métodos de la librería Game Kit utilizados por si solos no cubren el correcto funcionamiento de este

tipo de arquitectura y, por lo tanto, se debe definir una serie de pasos para poder completar cada una de las diferentes tareas que necesita realizar la red.

Estas tareas en que se necesita definir un protocolo son las siguientes: incorporación de un nuevo dispositivo a la red, envío - recepción de paquetes, desconexión de un dispositivo de la red y la reconexión de un dispositivo.

4.6.3. Protocolo de incorporación de un nuevo dispositivo a la red

Posiblemente es fácil pensar que no hay necesidad de crear un protocolo para solamente incorporar un dispositivo más a la red y que, inmediatamente después de conectarse, se podrá realizar las comunicaciones de la aplicación sin ningún problema. Por desgracia, la realidad es muy diferente porque durante la incorporación de un nuevo dispositivo hay retardos en el descubrimiento del nuevo miembro por parte de los demás dispositivos de la red y, además, hay la necesidad de enviar una serie de datos relacionados con la red antes de poder iniciar la actividad normal de comunicación de la aplicación.

Para solucionar los problemas descritos anteriormente será necesario seguir los siguientes pasos.

Imagen 4.10

La imagen 4.10 muestra por una parte la situación normal en que siempre se encuentra una red y, por otra, el intento de un dispositivo a unirse a una red. Cada dispositivo de la imagen muestra el estado en que se encuentra.

En la red se muestra dos tipos de estados, el servidor y el peer. El dispositivo que tiene el estado **servidor** es aquel tiene la misión de recibir los intentos de conexión a la red debido a que este está al final de la cadena de conexión. En cambio, los dispositivos que tiene el estado de **peer** son aquellos que ya tienen los vecinos establecidos (exceptuando el primero que fue el creador de la red) y su papel es la de mantener el comportamiento normal de la red. El estado Servidor se puede interpretar como un peer que tiene una función más ya que su papel en los demás protocolos es la misma.

El dispositivo con el estado **Buscador** tiene la misión de buscar todos aquellos dispositivos que hacen de Servidor. Si el Buscador encuentra más de un dispositivo eso significará que hay más de una red disponible para conectarse y en ese caso la Aplicación (o usuario de la Aplicación) deberá escoger uno.

Imagen 4.11

Una vez realizada la conexión del Buscador con el Servidor, los demás dispositivos que están en estado Peer se percataran que hay un nuevo dispositivo conectado a la red, tal como se puede observa en la imagen 4.11. El Buscador necesita enviar a todos los miembros de la red un primer paquete con información sobre él. El problema surge cuando el Buscador quiere enviar el paquete inmediatamente después de conectarse, ya que es muy posible que no todos los peers se hayan percatado de su presencia y, por lo tanto, estos no recibirán el paquete.

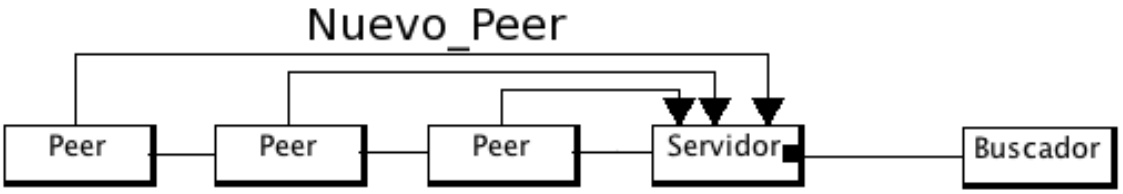

Imagen 4.12

Para solucionar este problema, el Buscador deberá espera a que todos los peers se hayan percatado de su incorporación a la red. Cada dispositivo que esté en estado Peer tendrá que enviar el paquete “Nuevo_Peer” al Servidor cuando estos se percaten de la nueva incorporación, tal como se ve en la imagen 4.12.

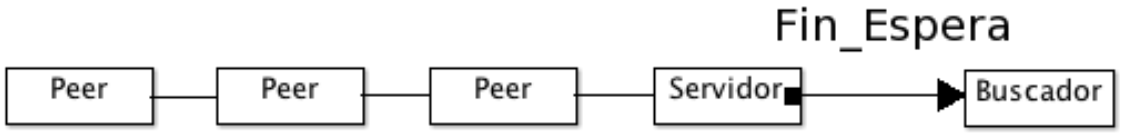

Imagen 4.13

Una vez que todos los peers indican al Servidor que han notado la incorporación del Buscador, el Servidor envía el paquete “Fin_Espera” al Buscador para indicarle que ya puede enviar el paquete con su información.

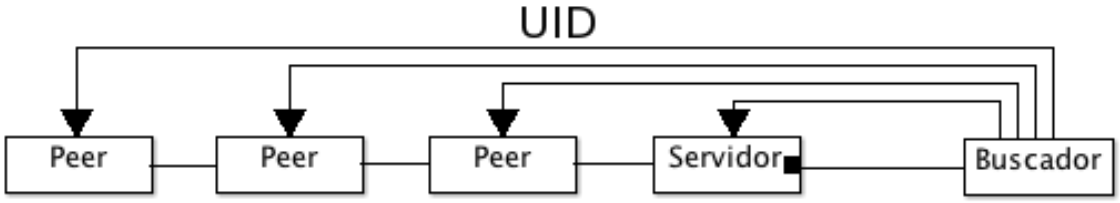

Imagen 4.14

En la imagen 4.14 se ve el envío del paquete “UID” (paquete que contiene información de Buscador) justo después de recibir el paquete “Fin_Espera”. Cada receptor del paquete registrará la información para actualizar el listado de dispositivos conectados.

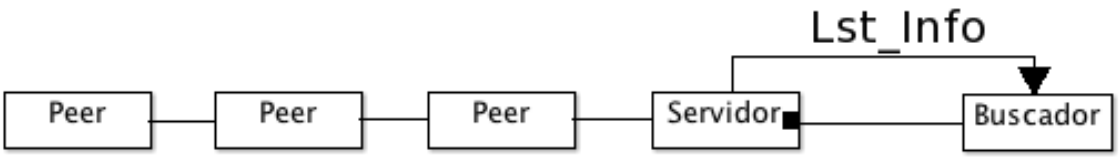

Imagen 4.15

Después que el Servidor reciba el paquete “UID”, él enviará un paquete al Buscador con una lista de datos, de todos los dispositivos de la red, que se ha ido generando a medida que se han unido nuevos dispositivos.

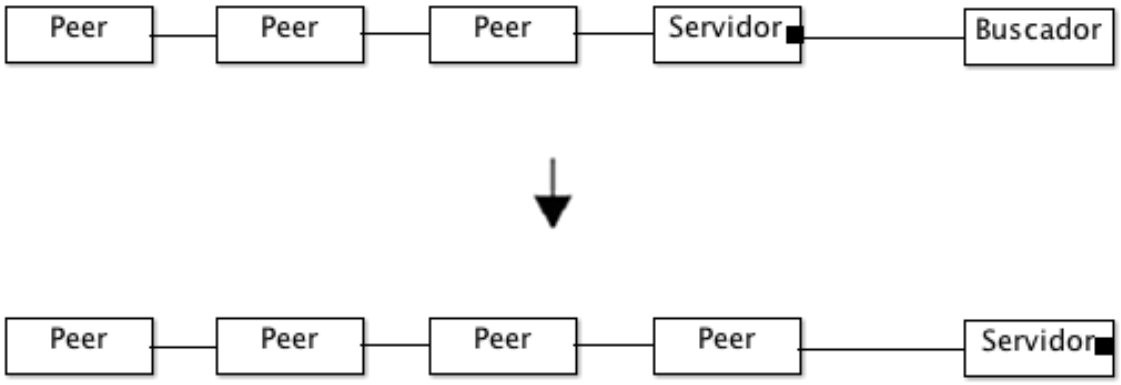

Imagen 4.16

Una vez que los dispositivos con estado Peer reciban el paquete “UID”, que el dispositivo Servidor envía el paquete “Lst_Info” y que el dispositivo Buscador reciba el paquete “Lst_Info”, los tres tipos de dispositivos realizarán una actualización de sus estados para finalizar el protocolo de incorporación de un nuevo dispositivo a la red. Los dispositivos con Estado Peer no cambian de estado, el dispositivo con estado Servidor cambia a estado Peer y el dispositivo con estado Buscador cambia y será el nuevo Servidor.

4.6.4. Protocolo de envío – recepción de paquetes

El intercambio de datos es una funcionalidad que esta cubierta con la utilización de dos métodos (uno que envía a un solo dispositivo y otro que envía a todos) y de un control de errores de envío. El control de errores tiene la misión de reenviar el paquete en caso que el receptor no lo haya recibido y si después de varios reenvíos el receptor no lo recibe, el control de errores muestra un error de envío. El problema del sistema de control de errores es que prueba demasiadas veces el

reenvío del paquete, provocando la interrupción de los envíos. Para solucionar este problema se ha decidido crear un protocolo para la confirmación de la recepción de los paquetes.

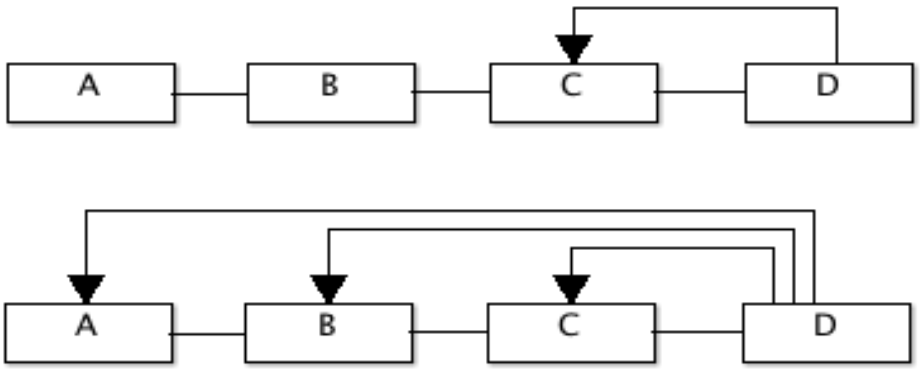

Imagen 4.17

El envío de los paquetes, tanto si a un destinatario como varios, se realizará de forma normal. Justo después se activará un temporizador por cada uno de los destinatarios de los paquetes.

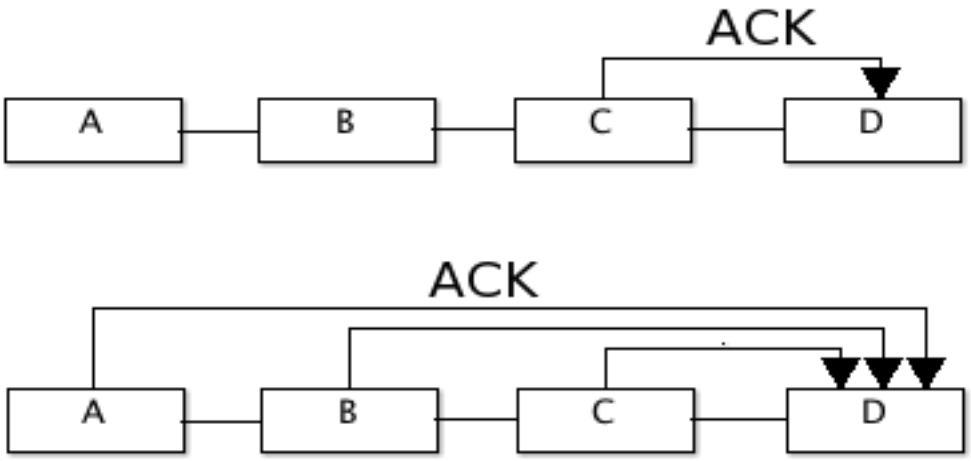

Imagen 4.18

Los receptores de un paquete deberán enviar un paquete “ACK” para contestar a su emisor, haciendo que él desactive el temporizador. Si en un cierto tiempo el emisor del paquete original no recibe un paquete “ACK”, el temporizador se activará y enviará un evento de error a la aplicación o usuario.

4.6.5. Protocolo de desconexión de un dispositivo de la red

Desconexión

La desconexión de un dispositivo es el momento más crucial del ciclo de vida de una red con la arquitectura en cadena, ya que muy posiblemente la red quedará dividida en dos subredes. La desconexión de uno de los dispositivos siempre tendrá el resultado de la división de la red en dos parte salvo en los casos que los desconectados sean los dispositivos que hacen de extremo (el dispositivo que creo la red y el que está en estado Servidor).

Imagen 4.19

La imagen 4.19 se puede observa una situación de desconexión del dispositivo que esta justo al centro de la cadena de conexiones que forma la red. Todos los dispositivo se han dado cuenta que el tercer dispositivo de la cadena se ha desconectado. Los dos primeros dispositivos de la cadena no solo se han dado cuenta de la desconexión del dispositivo del medio, sino que también han percibido la desconexión del cuarto y del quinto dispositivo debido a que ahora no están unidos por la cadena de conexiones y viceversa con el cuarto y el quinto respecto al primero y el segundo.

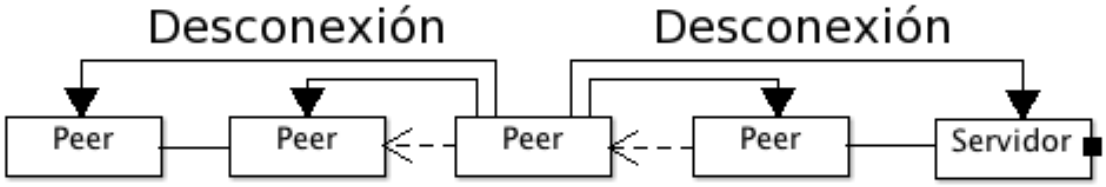

Imagen 4.20

En el caso que la desconexión del tercer dispositivo fuera intencionada como por ejemplo el cierre de la aplicación por parte del usuario, la recepción de una llamada, etc; el tercer dispositivo enviaría un paquete indicando a los demás dispositivo su intención de desconectarse. Los demás dispositivos, al recibir este paquete, sabrán que, en el momento de realizar la búsqueda del dispositivo que vuelva a unir la red, éste no será el tercer dispositivo.

Búsqueda

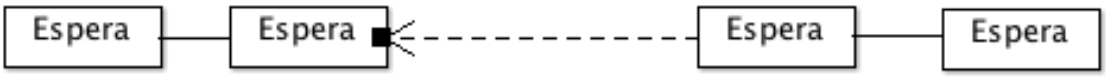

Imagen 4.21

En la imagen 4.21 se puede observar la siguiente etapa después de la desconexión. Todos los dispositivos han cambiado su estado a Espera. El estado Espera indica que el funcionamiento normal de las comunicaciones han sido interrumpidas y que se está realizando el proceso de unión o recuperación de la red. El dispositivo que estaba en estado Servidor, antes de cambiar de estado, ha cerrado la entrada a la incorporación de nuevos dispositivos para simplificar el proceso.

El proceso de recuperación esta divido en dos partes, la subred que espera la petición de unión y la subred que debe buscar a la otra para realizar la petición de unión. Los dispositivos que eran vecinos del dispositivo que se ha desconectado son los que tienen la misión de realizar la unión.

Imagen 4.22

El dispositivo que era el vecino izquierdo y que pertenece a la subred que espera la reconexión, esperará durante un tiempo que uno de los dispositivos que pertenece a la otra se conecte a él. El tiempo de espera variará según el número de dispositivos haya en la otra subred.

Imagen 4.23

El dispositivo que era el vecino derecho del dispositivo que se ha desconectado y que pertenece a la subred que debe conectarse con la otra, buscará durante un tiempo al dispositivo más cercano de la parte izquierda de la cadena de conexiones (que son aquellos que están en la subred que esta esperando la unión). Si durante un tiempo no encuentra el vecino izquierdo más cercano, el dispositivo procederá a buscar al siguiente vecino por la parte izquierda y así continuará buscando de la misma manera hasta llegue al inicio de la cadena. Si el proceso de búsqueda fuera buscar cualquier dispositivo de la otra subred podría pasar que, en el caso que la otra subred tuviera una desconexión, se encontrara un dispositivo que perteneciera a una sub-subred.

Imagen 4.24

Si en las subred se detecta nuevas desconexiones, el protocolo realizaría el mismo proceso descrito en la subred afectada y así de forma concurrente.

En el caso que no se produjese ninguna conexión de unión de las subredes, las respectivas subredes pasarían a la etapa de “espera de reconexión” que se describirá más adelante.

Comprobación

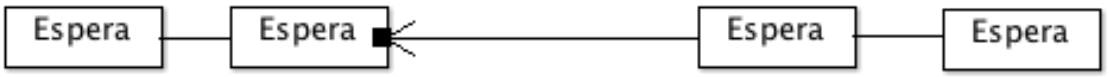

Imagen 4.25

Una vez realizada la conexión de los dispositivos que representan cada una de las dos subredes, se debe iniciar un proceso de comprobación para asegurase que ningún otro dispositivo esté aun buscando o esperando unirse con alguna otra subred debido a nuevas desconexiones.

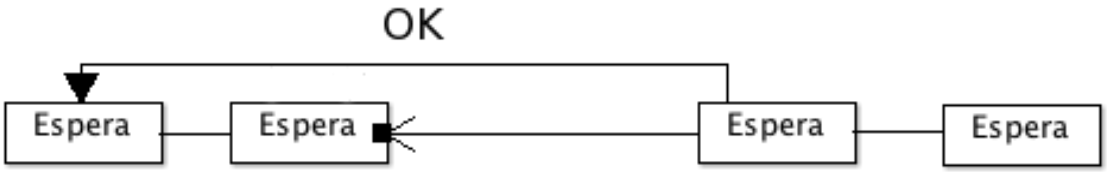

Imagen 4.26

La imagen 4.26 muestra el inicio del proceso de comprobación, el dispositivo que realizo la conexión entre las dos subredes envía el paquete “OK” al vecino izquierdo más alejado (el primero de la unión de las dos subredes) para indicarle que inicie la comprobación.

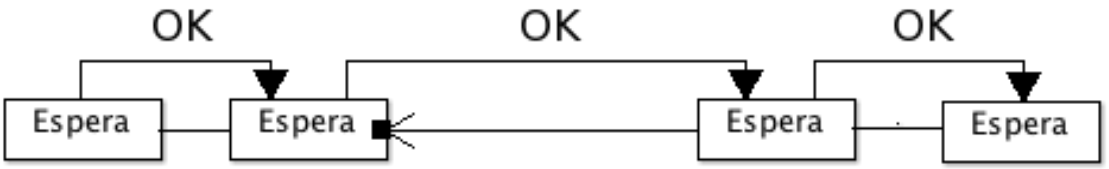

Imagen 4.27

El primer paso de la comprobación es iniciar una cadena de paquetes que llegue hasta el otro extremo de la red. El receptor del paquete “OK” comprueba que no tiene pendiente la unión de ninguna subred y continua la cadena con el envío del paquete a su vecino derecho. Si el receptor del paquete se encuentra en medio de una desconexión, tal como se describió en el paso anterior, no continuará la cadena de envíos, deteniendo así el proceso.

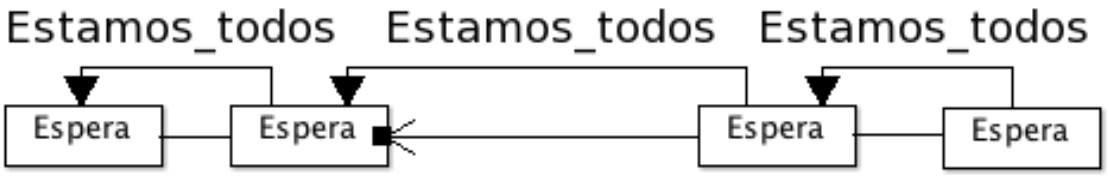

Imagen 4.28

El segundo paso de la comprobación se inicia al final de la cadena o extremo derecho de la red y también realizará otra cadena de paquetes que llegue hasta el otro extremo de la red. El dispositivo que inicia este proceso (el ultimo dispositivo de la red) enviará a su vecino izquierdo el paquete “Estamos_todos” añadiendo el número de dispositivos que reconoce están conectados en la red. El receptor del paquete “Estamos_todos” comprobará que el número de dispositivos conectados que lleva el paquete coincide con el suyo y continuará la cadena reenviando el paquete a su vecino izquierdo. Si un receptor encuentra que el número de dispositivos conectados son diferentes, no continuará la cadena de reenvíos, deteniendo así el proceso.

La razón de que el número de conectados no sea igual en todos los dispositivos, en el momento de iniciar el proceso de comprobación, es porque en algunos casos el inicio del proceso ha sido tan rápido que algunos dispositivos aun no han detectado a todos los dispositivos conectados en la red.

Si en uno de los pasos anteriores se ha detenido el proceso de comprobación esté volverá iniciarse en el momento que haya: una nueva unión de subredes o cuando un dispositivo detecte la reconexión de cualquiera de los dispositivos que había detectado como desconectado (esto pasa justo después que de la unión de subredes).

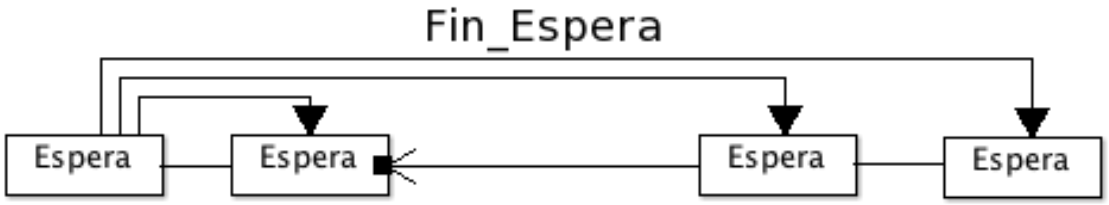

Imagen 4.29

Después de comprobar que todos los dispositivos tienen el mismo número de conectados, el primer dispositivo de la red envía a todos el paquete “Fin_Espera” para indicarles el inicio de la etapa “espera de reconexión”.

Espera de reconexión

Imagen 4.30

Una vez que cada dispositivo recibe el paquete “Fin_Espera” y que el primero envía el paquete, todos cambian su estado (Espera) a Espera_Reconexión. El único dispositivo que no realizará el mismo cambio de estado será el ultimo de la red, que su estado cambiará a Espera_Reconexión_Servidor. En el protocolo de reconexión, que vendrá a continuación, se explicará con más profundidad el papel de estos nuevos estados.

4.6.6. Protocolo de reconexión de un dispositivo

El protocolo de reconexión de un dispositivo actúa cuando todos los dispositivos de la red se encuentran en el estado Espera_Reconexión, tal como se puede ver en la imagen 4.30.

El estado Espera_Reconexión es el estado en que la red aun no ha vuelto a su actividad normal y espera a que alguno de los dispositivos que se desconectaron vuelvan a reconectarse a la red. El estado Espera_Reconexión es un estado muy similar al del Peer, tanto en la asignación de un estado especial para el último dispositivo de la red (por ser el punto donde se realiza las nuevas conexiones), como el funcionamiento del protocolo de incorporación.

El protocolo de reconexión es casi idéntico al protocolo de incorporación de un nuevo dispositivo a excepción de algunos puntos.

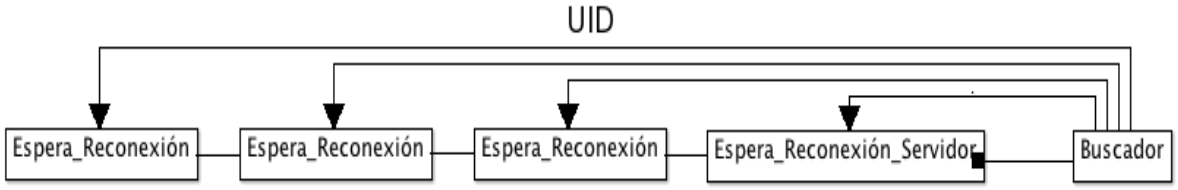

Imagen 4.31

La primera diferencia consiste en el comportamiento de los dispositivos de la red al recibir el paquete “UID”, tal como se puede ver en la imagen 4.31. Si el identificador que contiene el paquete es uno que no pertenece a ninguno de los dispositivos que se desconectaron, se rechazará la conexión de la red con el dispositivo Buscador. En el estado Espera_Reconexión solo aceptará conexiones de dispositivos que se desconectaron y nunca se aceptará la incorporación de nuevos dispositivos.

Imagen 4.32

La segunda diferencia es en el envío del paquete “Lst_Info”. En vez de enviar este paquete se le envía el paquete “Lst_Info_Re”. Los dos paquetes contienen la mismas información, pero el “Lst_Info_Re” avisa al dispositivo Buscador que esta realizando una reconexión como dispositivo que anteriormente pertenecía a la red.

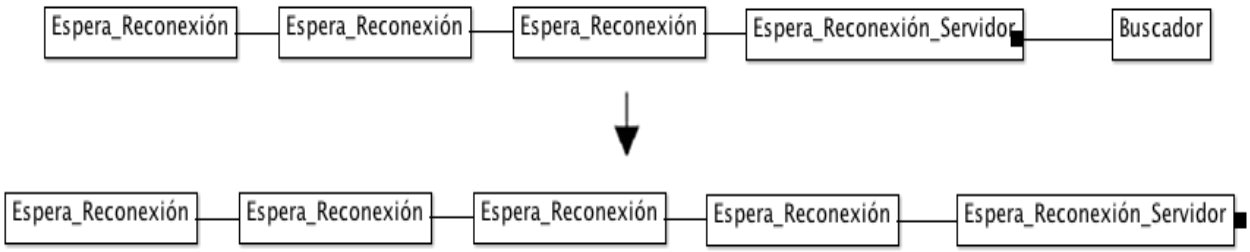

Imagen 4.33

Después de recibir el paquete “Lst_Info_Re” y tal como se hacia en el protocolo de incorporación, los dispositivos con estado Espera_Reconexión_Servidor y Buscador cambian sus estados, confirmando así la reconexión de un dispositivo que estaba desconectado.

Si durante el estado Esperando_Reconexión se sufre alguna desconexión se procedería igual que cuando se está en estado Peer y siguiendo el protocolo de Desconexión. Naturalmente la desconexión implicaría cambiar del estado Espera_Reconexión al estado Espera.

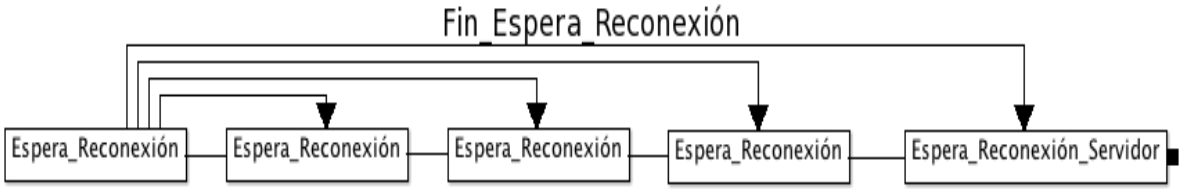

Imagen 4.34

Cuando todos los dispositivos que se habían desconectado estén reconectados o cuando el usuario del primer dispositivo haya decido que no habrá más reconexiones, el primer dispositivo enviará a todos los dispositivos de la red el paquete “Fin_Espera_Reconexión”.

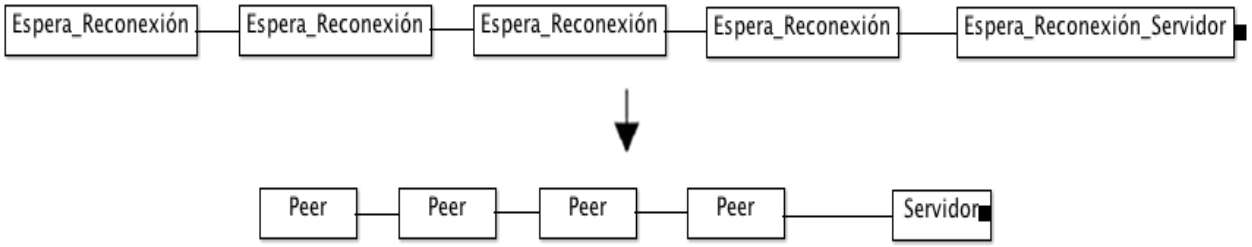

Imagen 4.35

Una vez que todos los dispositivos hayan recibido el paquete “Fin_Espera_Reconexión”, todos cambiarán el estado de Espera_Reconexión a Peer. En el caso del ultimo dispositivo el cambio de estado será de Espera_Reconexión_Servidor a Servidor. Este último paso está representado en la imagen 4.35.

Con este último paso, la red ya ha vuelto a la normalidad y esta lista para volver a iniciar las comunicaciones de forma normal.

Antes de dar paso al siguiente apartado, hay que remarcar que gracias al protocolo de reconexión siempre habrá la oportunidad de rehacer la red, a pesar que haya habido problemas de conexión o incluso un error en la aplicación. Esto siempre será posible si al menos un dispositivo no sale de la aplicación que utilice esta librería.

4.6.7. Diagrama de estados

A continuación se va a presentar el diagrama de estados que resume todos los cambios de estado que se han ido viendo en los diferentes protocolos.

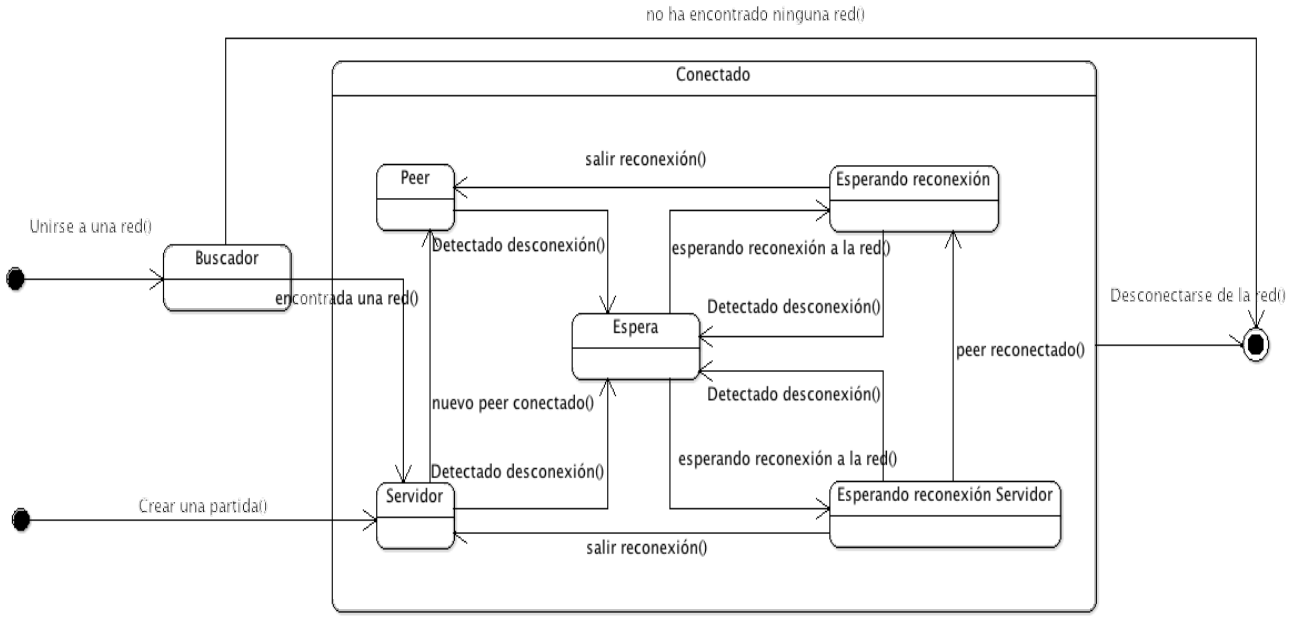

Imagen 4.36

4.6.8. Arquitectura en capas

La arquitectura que se va seguir para diseñar la librería de comunicación es la arquitectura en capas. La arquitectura en capas tiene el objetivo de separar todo el código del sistema en diferentes capas según un tipo de lógica de trabajo del sistema como por el ejemplo la lógica de negocio o la lógica de diseño – dominio. Otra interpretación de la división en capas es según la naturaleza del código, como por ejemplo el código que trabaja en bajo nivel creando funcionalidades básicas y el código de alto nivel que trabaja con muchas llamadas a funcionalidades básicas.

Una conocida reinterpretación de esta arquitectura es el patrón modelo vista controlador visto en el tema 3, apartado 3.3.1, donde dividen el código en tres capas. La capa de vista controla todo el código de la presentación y la iteración del usuario con la aplicación. La capa modelo contiene el código de todos los objetos del dominio del sistema. Y la capa controlador es la capa intermedia entre las dos anteriores y que contiene toda la lógica de negocio de la aplicación.

Las ventajas de utilizar la arquitectura en capas son varias: **abstracción** entre capas que permiten la organización del trabajo en diferentes equipos – capas, **cambiabilidad** de una capa por otra con las mismas funcionalidades pero implementadas de otra manera, **reusabilidad** de la capa en otras aplicaciones y **portabilidad** para utilizar la capa en otros sistemas.

Las desventajas de esta arquitectura son: poca **eficiencia** debido a que para hacer una funcionalidad tiene que pasar por muchas capas, a veces la faena hecha en una capa es **innecesaria o redundante** y la **complejidad** por el número de granularidad y de capas utilizadas.

Desarrollo

Para este desarrollo se ha decidido utilizar una arquitectura en **dos** capas. Una capa controlará la lógica de negocio, explicada a partir de los diferentes protocolos en apartado anteriores, y la recepción de los diferentes eventos de la clase GKSession (framework Game Kit). Y la segunda capa es la capa de dominio que esta controlado por un controlador que gestiona la información de los diferentes dispositivos conectados, la lista que representa la cadena de conexión, el control de los dispositivos que hacen de vecinos de éste, etc.

A diferencia del desarrollo normal, como por ejemplo el de una aplicación, el desarrollo de una librería no necesita las capas de presentación o de gestión de base de datos porque serán implementadas por el desarrollador que utilice la librería. Si se mira desde la perspectiva del desarrollador de una aplicación que utiliza la librería, ésta se puede considerar una capa más (la capa de comunicación) del desarrollo de una aplicación.

En unos apartados más adelante se presentará el diagrama de clases con la división de las dos capas.

4.6.9. Patrón de diseño – El patrón fachada

En el diseño de software es muy común la utilización de patrones de diseño para resolver diferentes problemas de diseño que el desarrollador se puede encontrar.

En este caso de desarrollo, el problema que se encuentra es el siguiente: el desarrollador tiene acceso a todas las clases y métodos del sistema, haciendo que la utilización del sistema sea un poco compleja o confusa al tener la necesidad de conocer todas las clases y los métodos.

El patrón que se ha decido utilizar para solucionar estos problemas es el patrón fachada.

La solución que sugiere el patrón fachada es la utilización de una única interfaz de alto nivel para todo el sistema y ocultar todas las interfaces de bajo nivel de las clases que lo implementan. De esta manera, el desarrollador le será más fácil de utilizar el sistema (la librería) porque solo tendrá que aprender la utilización de una clase. Aprovechando de que se va hacer compilar el proyecto en una librería, las interfaces del sistema se declararán privadas, mientras que la interfaz que hace de fachada será pública.

En el diagrama de clases del siguiente apartado se podrá ver la aplicación del patrón Fachada.

4.6.10. Diagrama de clases

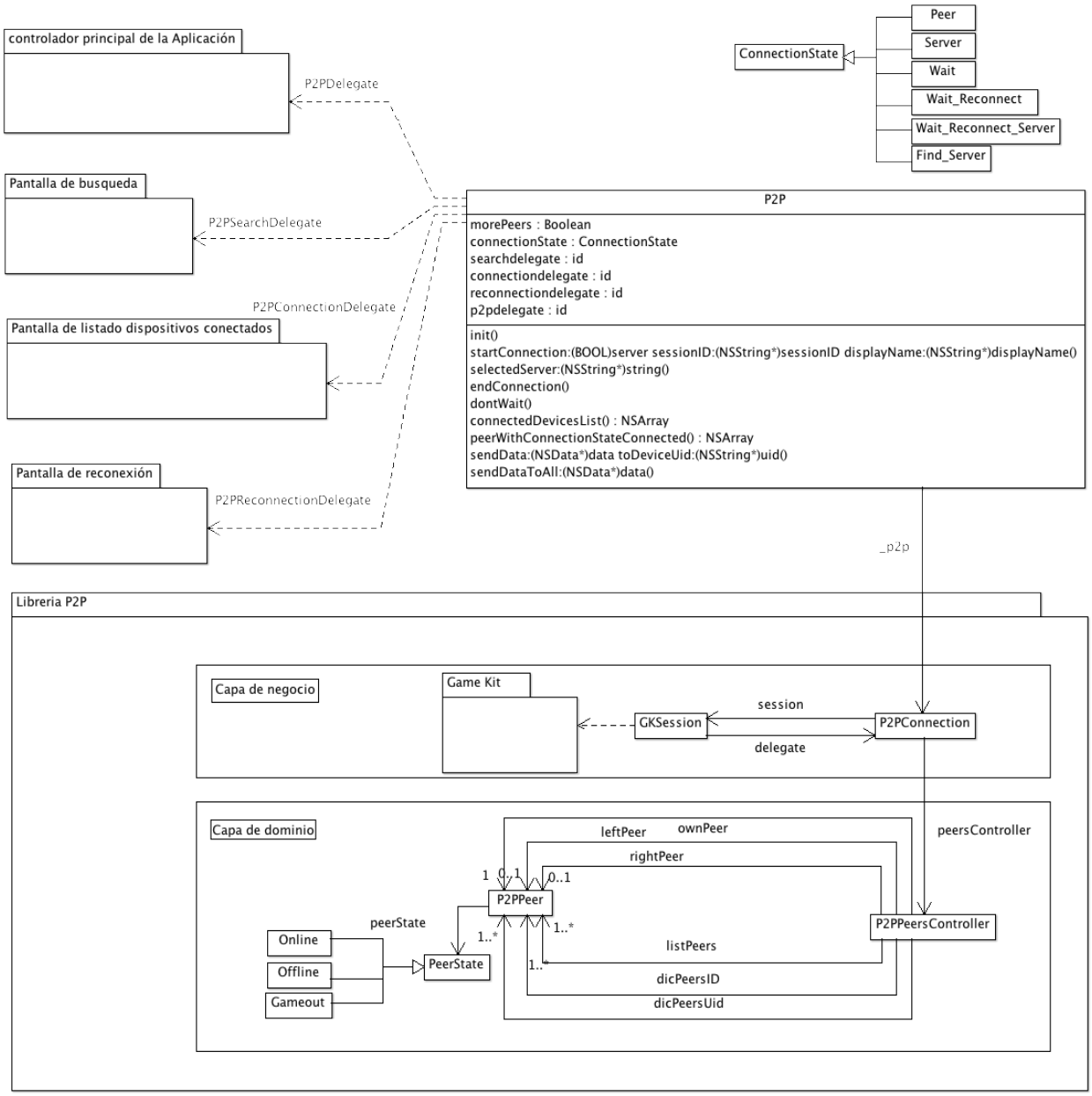

Imagen 4.37

El diagrama de clases presentado en la imagen 4.37 se puede ver dos partes, la clase P2P y la librería P2P con sus correspondientes clases. Para no complicar el diagrama, se ha preferido detallar solo los atributos y los métodos de la clase P2P.

La clase P2P será la única clase que el desarrollador de la aplicación (que utiliza la librería) podrá acceder, tal como se explicó en el apartado 4.6.9 (patrón fachada). Los atributos que sus nombres acaben en –delegate (delegados) son punteros a diferentes objetos que recibirán eventos (llamadas) que se vayan produciendo en la conexión y, tal como se explicó en el capítulo 3 apartado 3.1.5, el patrón delegado utiliza los protocolos para definir los métodos que el objeto apuntado debe tener implementados. Las flechas discontinuas que se ven en la izquierda del diagrama son los

diferentes protocolos por cada uno de los puntero delegados de P2P que apuntan a los diferentes objetos que en principio están pensados a recibirlos.

La librería P2P es la parte del diagrama de clases en que el desarrollador no podrá ni acceder ni ver. El número de clases que compone la librería es bastante pequeña, pero cambio el número de relaciones que hay entre ellas es algo mayor. Las clases de la librería esta divida en dos capas que ya han sido descritas en el apartado 4.6.8.

4.7. Codificación - implementación

A continuación se va a explicar algunos detalles relacionados con la etapa de implementación.

Para implementar el código de la librería se creado un proyecto nuevo con una plantilla predefinida del xcode llamada “Cocoa Touch Static Library”.

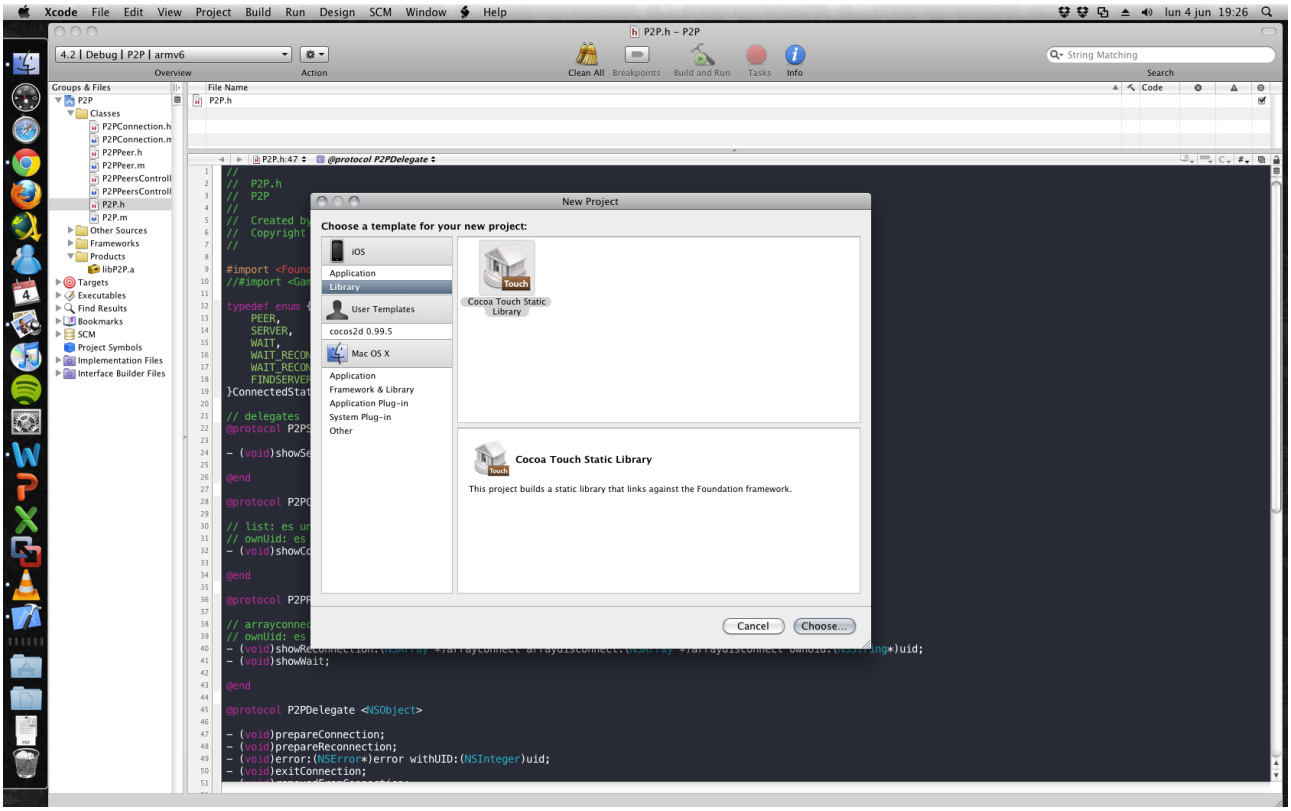

Imagen 4.38

Utilizando esta plantilla, el proyecto esta configurado automáticamente para compilar todas las clases a dentro de un archivo con extensión .a. Este archivo es la librería estática del proyecto y para utilizar las clase que contiene es necesario proveer, junto con la librería, los archivos de cabecera con extensión .h. En nuestro caso y tal como se ha explicado en el apartado 4.6.9., se quiere que solo sea accesible la clase P2P para que sea el único punto acceso para el desarrollador. El archivo de cabecera que se dará junto a la librería es el siguiente:

```
//
//  P2P.h
//  P2P
```

```
//

#import <Foundation/Foundation.h>
#import <GameKit/GameKit.h>

typedef enum {
      PEER,                                    // ONLINE.
      SERVER,                                  // ONLINE.
      WAIT,                                    // ONLINE.
      WAIT_RECONNECT,                          // ONLINE.
      WAIT_RECONNECT_SERVER,                   // ONLINE.
      FINDSERVER                               // OFFLINE.
}ConnectedState;

// delegates
@protocol P2PSearchDelegate <NSObject>

- (void)showSearch:(NSDictionary *)dic;

@end

@protocol P2PConnectionDelegate <NSObject>

// list: es un Array de Array (Matriz 2*m) con dos Strings (Nombre y uid)
// ownUid: es nuestro identificador unico, asi nos situamos dentro de list

- (void)showConnection:(NSArray *)list ownUid:(NSString*)uid;

@end

@protocol P2PReconnectionDelegate <NSObject>

// arrayconnect y arraydisconnect: es un Array de Array (Matriz 2*m) con dos Strings
(Nombre y uid)
// ownUid: es nuestro identificador unico, asi nos situamos dentro de list

 - (void)showReconnection:(NSArray *)arrayconnect
   arraydisconnect:(NSArray*)arraydisconnect ownUid:(NSString*)uid;
- (void)showWait;

@end

@protocol P2PDelegate <NSObject>

- (void)prepareConnection;
- (void)prepareReconnection;
- (void)error:(NSError*)error withUID:(NSInteger)uid;
- (void)exitConnection;
- (void)removedFromConnection;
- (void)receiveData:(NSData *)data fromUid:(NSString *)uid;
- (void)newDevicesConnected:(NSString*)uid name:(NSString*)name;
- (void)devicesDisconnected:(NSArray*)array;
- (void)devicesReconnected:(NSString*)uid;
@optional
- (void)startApplication;
@end

@interface P2P : NSObject {
      @private
      id                                                    _p2p;
}

@property(nonatomic)BOOL                                    morePeers;
@property(nonatomic, readonly)ConnectedState                connectionState;

// delegates
@property(nonatomic, assign)id<P2PSearchDelegate>           searchdelegate;
@property(nonatomic, assign)id<P2PConnectionDelegate>       connectiondelegate;
@property(nonatomic, assign)id<P2PReconnectionDelegate>     reconnectiondelegate;
@property(nonatomic, assign)id<P2PDelegate>                 p2pdelegate;
```

```
- init;
- (void)endConnection;
- (void)startConnection:(BOOL)server sessionID:(NSString*)sessionID
displayName:(NSString*)displayName;
- (void)selectedServer:(NSString*)string;
- (void)dontWait;
- (NSArray*)connectedDevicesList;
- (NSArray*) peerWithConnectionStateConnected;
- (BOOL)sendData:(NSData*)data toDeviceUid:(NSString*)uid;
- (void)sendDataToAll:(NSData*)data;

@end
```

4.8. Pruebas

Para realizar las pruebas de la librería se ha seguido las indicaciones del apartado 4.2.3 y se ha implementado una pequeña aplicación que, utilizando la librería, realiza el envío de strings a todos los dispositivos de la red.

Las pruebas que se han realizado para comprobar el correcto funcionamiento de la librería son las siguientes:

- Intento de incorporación de varios dispositivos, a la vez, a un único dispositivo (haciendo de servidor). El resultado de la prueba es que uno de los dispositivos se ha unido a la red y los demás dispositivos, que intentaban unirse, han sido rechazados con la llegada de un evento del tipo aviso.
- Intento de incorporación de un dispositivo a una red cuando ésta ha sufrido una desconexión y está en proceso de recuperación de la red. El resultado de la prueba es el rechazo de este dispositivo debido a que éste no se le reconoce como dispositivo que perteneciera a la red y recibe un evento del tipo aviso.
- Desconexiones múltiples de dispositivos que pertenecen a la red. Esta prueba se ha intentado realizar varias veces en diferentes posiciones de la red: los dispositivos que están en los extremos, los dispositivos que están en el centro, dispositivos que están en posiciones par y dispositivos que están en posiciones impar. Los resultados han sido la recuperación de la estructura de la red y con la posibilidad de reincorporar los dispositivos desconectados. Muy raramente ha pasado que algún dispositivo no encontrará algún otro para realizar la recuperación de la red, pero con el reinicio de la aplicación y con que los otros dispositivos estando a la espera de reconexiones de dispositivos desconectados, se ha vuelto a recuperar la red.
- Las mismas pruebas del punto anterior pero estando la red parada en el estado de espera de reconexión de dispositivos que se habían desconectado. Se ha obtenido los mismo resultados que el punto anterior.
- Cada dispositivo tiene que enviar a todos los demás de la red un string cada segundo. El resultado ha sido que no ha habido problemas de envío y de recepción de los paquetes.

Además de comprobar por la pantalla del dispositivo el funcionamiento de las pruebas, también se ha comprobado con la herramienta de consola, que está integrada en el xcode, unos datos que indican como esta la estructura de la red.

<Notice>: ====== 4 nodes =====

<Notice>: 0: 2020A919 Next[2020A919], dist[0], SN[1], New[0], From[2020A919], #NEIGHBOR[1] "Dispositivo A"[5]

<Notice>: 5FE48D2C, RTT[644]

<Notice>: 1: 5FE48D2C Next[5FE48D2C], dist[644], SN[2], New[0], From[5FE48D2C], #NEIGHBOR[2] "Dispositivo B"[18]

<Notice>: 2020A919, RTT[424]

<Notice>: 6AE28B8C, RTT[389]

<Notice>: 2: 6AE28B8C Next[5FE48D2C], dist[1033], SN[2], New[0], From[5FE48D2C], #NEIGHBOR[2] "Dispositivo C"[15]

<Notice>: 5FE48D2C, RTT[170]

<Notice>: 7A060463, RTT[878]

<Notice>: 3: 7A060463 Next[5FE48D2C], dist[1911], SN[1], New[0], From[5FE48D2C], #NEIGHBOR[1] "Dispositivo D"[15]

<Notice>: 6AE28B8C, RTT[1326]

En la consola se puede apreciar el número de enlaces que mantiene cada dispositivo, teniendo dos los dispositivos centrales y solo uno los dispositivos de los extremos. Por desgracia esta información solo los mostraban los dispositivos con la tercera versión del iOS, siendo eliminada a partir de la cuarta versión.

4.9. Limites y filosofía de la librería

Limites

A pesar de que la librería ha sido diseñada para superar ciertos problemas o carencias detectadas de la librería Game Kit, ésta también tiene una serie limitaciones o problemas.

La arquitectura de la red y los protocolos diseñados están pensados para no tener problemas de número de dispositivos conectados, pero estos no tienen en cuenta los envíos de paquete cuando son muchos dispositivos. El problema radica en que si un dispositivo tiene que enviar un paquete a todos los dispositivos de la red y esta red es muy grande, es posible se produzca un desfase en la recepción del paquete de los primeros dispositivos respecto a los últimos dispositivos. Este problema es un problema teórico porque la prueba con dispositivos reales más grande que se ha podido hacer ha sido con cuatro dispositivos y estos no tuvieron este problema. La solución a este problema es el diseño de un protocolo de sincronización, en la librería o incluso en la aplicación que use la librería, para conseguir el efecto de que todos los dispositivos reciben a la vez el mismo paquete.

Otro problema que se ha detectado es el funcionamiento del Bluetooth en algunos dispositivos es anómalo, con la consecuencia de provocar la desconexión del Bluetooth. Si creamos redes muy grandes es muy posible que encontremos más fácilmente dispositivos con este comportamiento, obtenido así más posibilidades de desconexiones fortuitas.

Filosofía

Es importante que el desarrollador de la aplicación que vaya a utilizar la librería diseñe la aplicación pensado que cada dispositivo de la red debe ser igual entre si, conteniendo los mismos datos y lógica de negocio. Siguiendo esta filosofía, el desarrollador conseguirá que la aplicación no sufra ningún problema si hay desconexiones en las comunicaciones. Por lo tanto, el desarrollador debe evitar el diseño de la aplicación pensado que hay un dispositivo que siempre será el servidor o almacenador único de datos.

Durante todo el desarrollo no se ha planteado nada sobre la seguridad de la red debido a que esta responsabilidad se ha delegado al desarrollador de la aplicación que vaya utilizar la librería. Entonces, la elección de utilizar, por ejemplo, un criptado para los paquete enviados estará en manos del desarrollador, según sus necesidades al momento de diseñar la aplicación.

5. JUEGO – PÓKER

En los capítulos anteriores se ha podido profundizado en el lenguaje Objective-c y en el desarrollo de aplicaciones en iOS. Parte de ese conocimiento adquirido se ha aplicado junto a la librería Game Kit para desarrollar la librería de comunicación peer-to-peer, tal como se ha visto en el capítulo anterior. La finalización del capitulo anterior alcanza el primer objetivo del proyecto y a lo largo de los dos siguientes capítulos (Juego – Poker y Cocos2d – la interfaz de usuario), se resolverá el segundo objetivo del proyecto.

Nota importante de este capítulo

Debido a que muchos apartados de este capítulo son iguales a los que se han visto en el anterior, se ha decidido omitir la introducción teórica del contenido de cada apartado para evitar repeticiones innecesarias. Si hay alguna duda sobre el objetivo de un apartado, solamente hay que volver al capítulo anterior y buscar el respectivo apartado.

5.1. Introducción

Una vez acabado el desarrollo de la librería de comunicación peer-to-peer es necesario probar su correcto funcionamiento y fácil utilización. La mejor manera de probarlo es utilizarlo en una situación real que ayude a comprobar su viabilidad. Por lo tanto, será necesario el desarrollo de una aplicación que necesite para su funcionamiento la comunicación entre varios dispositivos.

Para este segundo desarrollo, se ha escogido el tipo de aplicación que se debía desarrollar, un juego. El juego en concreto debe ser uno que necesite la participación de varios jugadores y que sea medianamente sencillo, con reglas claras y conocidas. Los primeros nombres que surgieron fueron el parchís, la oca, el uno, la brisca, el tute, el mus, etc. Entre ellos, hubo uno que llamó más la atención, el póker.

El póker es el juego de cartas más famoso del mundo. El juego tiene multitud de variantes, pero la más conocida y jugada hoy en día es el Texas Hold'em.

Una vez escogida la aplicación sobre la cual se hará el caso de estudio, se procederá al desarrollo que se documenta a continuación.

5.2. Metodología de trabajo

Este segundo desarrollo, que tiene el fin de cumplir el segundo objetivo del proyecto, seguirá también la metodología de desarrollo en cascada, o ciclo de vida clásico, visto en el apartado 4.4.1. del capítulo anterior.

Al igual que pasó en el capítulo anterior, se ha decido utilizar esta metodología de desarrollo porque el juego escogido tiene una reglas bien definidas y porque el desarrollador ya tiene

experiencia previa en el juego. Debido a que el desarrollador ya ha jugado al juego con anterioridad, se le puede considerar como un usuario o un cliente también, facilitando aun más el proceso de especificación.

5.3. Especificación

5.3.1. Definición del sistema

El póker

El póker es un juego de apuestas donde los jugadores utilizan las cartas de un mazo, previamente repartidas, como medio para definir sus apuestas.

Las cartas repartidas a cada jugador son cartas ocultas que los demás jugadores no pueden saber su valor. Según la variante del juego, es posible que también hayan otras cartas que sean descubiertas y visibles para todos los jugadores. El tipo de cartas utilizadas comúnmente para jugar al póker es el inglés, formado por 52 cartas. El mazo inglés está formado por 4 palos (Diamantes, Picas, Corazones y Tréboles) y cada palo tiene un rango de 13 valores, formado por 4 cartas literales y 9 numerales (el orden de mayor valor a menor es A, K, Q, J,10, 9, 8, 7, 6, 5, 4, 3 y 2).

El objetivo del juego es ganar el bote de cada ronda mediante un mecanismo de apuestas. Cada jugador empieza con los mismos puntos y el juego acaba cuando un jugador consigue todos los puntos que hay en juego.

Cada mano (o ronda) se mezclará todas las cartas del mazó y se repartirán un número fijo de cartas a cada jugador. A partir de esas cartas repartidas, los jugadores iniciarán el proceso de apuestas que tendrá como resultado un ganador o varios ganadores. El número de cartas a repartir varía según la variante del póker que se juegue.

El proceso de apuestas consiste en un número fijo de rondas de apuestas determinado por la variante del juego. En cada ronda de apuestas, un jugador inicia su turno con una de estas tres acciones posibles: apostar, pasar o retirar.

- Apostar: esta acción consiste en jugar o poner un número de puntos (puede ser puntos, euros, dólares, etc.) para afirmar que sus cartas tienen valor. No siempre la realización de una apuesta conlleva este significado porque el jugador puede mentir o engañar a los demás jugadores, haciéndoles creer que tiene unas cartas que no tiene (farol). Existen 4 tipos de apuestas:
    - Apostar (Bet): se dice apostar, a secas, cuando anteriormente nadie aún no ha apostado nada durante esa ronda de apuestas.
    - Ver (Call): se apuesta la misma cantidad de puntos que la apuesta más alta de esa ronda de apuestas.

  - Aumentar (Raise): se apuesta una cantidad más alta que la actual apuesta más alta de esa ronda de apuestas.
  - Todo (All-in): se apuesta todos los puntos disponibles del jugador.
- **Pasar**: esta acción consiste en no realizar la acción de apostar y conceder el turno al siguiente jugador. Esta acción es posible si antes no se ha realizado ninguna apuesta durante esa ronda de apuestas, en caso contrario no se podrá hacer esta acción y solo se podrá apostar o retirarse.
- **Retirar**: esta acción consiste en desechar las cartas y dejar de participar en la actual mano. Todos los puntos que hubiera apostado el jugador, durante las anteriores rondas de apuestas del actual mano, los pierde.

Una vez que el jugador inicial realiza una acción, el turno pasa al siguiente jugador y así de forma consecutiva hasta que todos los jugadores hayan realizado una acción. Una vez hecha toda una vuelta, se debe mirar si hay más de un jugador que no se haya retirado y, si es así, mirar si todos los jugadores, que no se han retirado, tienen la misma cantidad de puntos apostados. En caso contario, se deberá iniciar una nueva vuelta con todos jugadores que aun no se han retirado y así hasta que todos tengan la misma apuesta o hasta que solo quede uno. Después de igualar todas las apuesta, habiendo más de un jugador, se iniciará una nueva ronda de apuestas, si es el caso.

Una mano acaba si en una de las rondas solo queda un jugador sin retirar o si se completa todas las rondas de apuestas. Si solo queda un jugador, este será el ganador del bote. Si se completa todas las rondas de apuestas, se procederá a la fase de destape de las cartas ocultas de cada jugador que opta al bote. En el destape gana el jugador que tenga la mejor combinación de 5 cartas. La clasificación de las combinaciones de 5 cartas, de mayor a menor valor, son las siguientes:

- **Escalera de color**: cinco cartas consecutivas del mismo palo. Si hay un empate, la escalera más alta gana. La escalera de color más alta que se puede conseguir se denomina escalera real y consiste en un as (A), rey (K), reina (Q), jota (j) y diez (10) del mismo palo. Una escalera real es una mano imbatible.

- **Poker**: cuatro cartas iguales y una diferente (kicker). Si hay un empate, el poker mayor gana. En juegos de cartas comunitarias donde los jugadores tienen el mismo poker, gana la quinta carta diferente más alta o kicker.

- Full house: tres cartas del mismo valor y dos cartas de un valor distinto, pero coincidentes entre ellas. Si hay un empate, las tres cartas del mismo valor más altas ganan el bote. En juegos de cartas comunitarias donde los jugadores tienen las mismas tres cartas iguales, gana la mano con las dos cartas de un valor distinto más altas.

- Color: cinco cartas del mismo palo. Si hay un empate, el jugador con la carta más alta gana. Si es necesario, la segunda, tercera, cuarta y quinta carta más alta pueden romper el empate. Si las cinco cartas tienen el mismo valor el bote se repartirá. En el juego de poker el palo de las cartas nunca se utiliza para deshacer un empate.

- Escalera: cinco cartas consecutivas. Si hay un empate, la escalera más alta gana. El as es la única carta que se puede usar en la parte alta o baja de la secuencia. La combinación A-K-Q-J-10 (Ace high) es la escalera más alta mientras 5-4-3-2-A (Five high) es la más baja.

- Trío: tres cartas del mismo valor y dos de un valor diferente. Si hay un empate, el trío más alto gana. En los juegos con cartas comunitarias donde más de un jugador posee el mismo trío, gana la carta más alta no emparejada, y si fuere necesario, la segunda carta más alta no emparejada.

- Doble pareja: dos cartas del mismo valor, combinadas con otras dos cartas del mismo valor y una quinta diferente. Si hay un empate, la pareja más alta gana. Si los jugadores tienen la misma pareja más alta, la segunda pareja más alta gana. Si ambos jugadores tienen parejas idénticas, gana la carta de valor diferente más alta.

- Pareja: dos cartas del mismo valor y tres de valores diferentes. Si hay un empate, la pareja más alta gana. Si los jugadores tienen la misma pareja, la carta diferente más alta gana y, si

es necesario, la segunda y la tercera carta diferentes pueden utilizarse para romper el empate.

- Carta alta: cualquier mano que no pertenece a ninguna de las categorías anteriores. Si hay un empate, la carta más alta gana y, si es necesario, la segunda, tercera y cuarta carta diferentes pueden utilizarse para romper el empate.

En caso de empate entre varios jugadores, el bote será divido a partes iguales.

El bote comunitario es la suma de todas las apuestas realizas por los jugadores durante una mano. En el caso de que un jugador haga un all-in y que los demás jugadores tenga más puntos que él, se creará otro bote o subbote (en total tenemos dos subbotes) que contendrá las apuestas superiores a la del all-in. El jugador del all-in solo podrá optar a ganar el primer subbote y no tendrá derecho sobre los siguientes subbotes que se creen. Por lo tanto, puede darse el caso de que haya varios ganadores de subbotes con jugadas diferentes.

El Texas Hold'em

Esta variante del póker sigue las misma reglas descritas anteriormente pero con algunas particularidades.

Se reparten a cada jugador dos cartas ocultas y se utilizan cinco cartas comunitarias que inicialmente no sé muestran.

Para garantizar un bote mínimo en cada mano, el Texas Hold'em introdujo el concepto de ciegas. Las ciegas son dos jugadores que debe realizar en la primera ronda de apuestas una apuesta mínima obligatoria que establece antes de iniciar la partida. Una ciega se llama "ciega pequeña" y la otra se llama "ciega grande". La ciega pequeña tiene que apostar la mitad de la apuesta mínima y la ciega grande tiene que apostar toda la apuesta mínima.

La apuesta mínima es la cantidad de puntos que se debe poner como mínimo para realizar una apuesta. Esta cantidad puede ser aumentada durante el transcurso de la partida, siendo lo normal doblar esta apuesta. El aumento de la apuesta mínima se realiza en el transcurso x minutos, pactados antes de iniciar la partida. El objetivo del aumento progresivo de la apuesta es acelerar la eliminación de jugadores y acortar el tiempo total de la partida.

Otro rol que también se utiliza en esta variante del póker es el repartidor. El repartidor (dealer) es el último jugador que tiene turno en una ronda de apuestas. Al ser el último, tiene una buena posición al momento de apostar porqué él podrá decidir si igualar la apuesta e iniciar una nueva ronda de apuestas, o aumentar la apuesta e iniciar una nueva vuelta.

El orden de estos tres roles son consecutivos. El jugador izquierdo del repartidor será la ciega pequeña y el siguiente jugador izquierdo, después de la ciega pequeña, será la ciega grande. Por lo tanto , la ciega pequeña será el primer jugador que tendrá turno en una ronda de apuestas y el segundo será la ciega grande. Si la partida se realiza con menos de tres jugadores, el jugador que hace de repartidor también será la ciega pequeña.

El número de rondas de apuestas que tiene el Texas Hold'em son cuatro: pre-flop, flop, turn y river.

De las cuatro rondas, la única que funciona diferente a las demás es la primera, el pre-flop. El pre-flop es la ronda donde se realiza las apuestas obligatorias de las ciegas, tal como se ha descrito en unas líneas más arriba. Además de eso, el primer turno de la ronda no lo comienza la ciega pequeña, tal como se haría normalmente, sino que lo inicia el siguiente jugador después de la ciega grande. El último jugador que cierra una vuelta en esta ronda corresponde a la ciega grande, que en esta ronda tendrá el mismo poder de decisión que tiene el repartidor en las demás rondas de apuestas. Aunque la ciega pequeña haya hecho una apuesta obligatoria en el pre-flop, tiene la obligación de aumentar su apuesta hasta como mínimo a la apuesta mínima si quiere continuar en las siguiente ronda.

Al iniciar las rondas de flop, turn y river se realiza el destape de las cartas comunitarias. Al inicio del flop se muestra las primeras tres cartas, en el turn la cuarta carta y en el river se enseña la quinta y última carta. En cada ronda, antes de sacar las cartas comunitarias del mazo, se realiza la quema de una carta. Quemar una carta es simplemente retirar una carta del juego o del mazo y no se muestra nunca que carta es. En el caso que en una ronda todos los jugadores que no se hayan retirado hacen all-in (o todos menos uno) se procederá directamente al destape de todas las cartas comunitarias y las de los jugadores implicados.

En el momento del destape de las cartas de los jugadores que optan al bote , las cartas que se utilizan para proceder a su clasificación son las cinco cartas con la mejor combinación posible. Cada jugadores tiene a su disposición siete cartas, las dos privadas y las cinco comunitarias, para realizar la mejor combinación de cinco cartas.

En el caso que la mejor combinación de cinco cartas posible son las cartas comunitarias, se procederá a repartir el bote entre todos los jugadores que no se hayan retirado.

El número máximo de jugadores que puede jugar el Texas Hold'em es de 22.

Otros aspectos

Se desea que los únicos jugadores que participen en el juego sean los usuarios de los dispositivos y que no vea ningún jugador no humano controlado por algoritmos de toma de decisión.

5.3.2. Análisis de requisitos

A continuación se va a describir cuales son las necesidades del sistema y ,tal como se explicó y se procedió en el capítulo anterior, se va dividir estos requisitos en dos grupos, requisitos funcionales y no funcionales.

En cada requisito analizaremos los siguientes parámetros:

- Requisito : identificador del requisito. RF, requisito funcional. RNF, requisito no funcional.
- Suceso / caso de uso : caso de uso en el cual se relaciona el requisito.
- Descripción : descripción del requisito.
- Justificación : motivo para cumplir este requisito.
- Dependencia : el requisito puede depender o requerir de otros requisitos del sistema.

Requisitos funcionales

Requisito: RF1	**Suceso / caso de uso:** Crear una partida nueva
Descripción: El sistema permitirá ejecutar una partida nueva.	
Justificación: El sistema permitirá al usuario lanzar una partida, que previamente ha sido configurada, y posteriormente permitirá que nuevos jugadores se una a ella.	
Dependencia: ---	

Requisito: RF2	**Suceso / caso de uso:** Crear una partida nueva
Descripción: El sistema permitirá configurar unos parámetros antes de lanzar una partida.	
Justificación: El sistema permitirá al usuario configurar parámetros relacionados con la partida creada como, por ejemplo, los puntos que tendrá cada jugador, la apuesta mínima, activar el sistema de auto aumento de las ciegas, etc.	
Dependencia: RF1	

Requisito: RF3	**Suceso / caso de uso:** Unirse a una partida
Descripción: Unirse a una partida no comenzada.	
Justificación: El sistema permitirá al usuario poder añadirse a una partida ya creada, pero aun no iniciada.	
Dependencia: RF1	

Requisito: RF4	Suceso / caso de uso: Crear partida / Unirse partida
Descripción: Iniciar la partida creada.	
Justificación: El sistema permitirá al usuario que creó la partida (o él que lleve más tiempo unido a la partida) iniciar el juego de póker. En ese momento no se aceptara la incorporación de más jugadores nuevos. No se podrá iniciar el juego si no hay como mínimo un total dos jugadores unidos a la partida.	
Dependencia: RF1, RF3	

Requisito: RF5	Suceso / caso de uso: Jugar
Descripción: Jugar al juego jóker Texas Hold'em.	
Justificación: El sistema permitirá a todos los usuarios unidos a la partida poder jugar al póker Texas Hold'em.	
Dependencia: RF1, RF3	

Requisito: RF6	Suceso / caso de uso: Jugar
Descripción: Interrumpir el juego por la ausencia de un jugador.	
Justificación: El sistema interrumpirá el juego cuando detecte que un jugador se ha desconectado, impidiendo a todos los jugadores realizar ninguna otra acción relacionada con el juego.	
Dependencia: RF5	

Requisito: RF7	Suceso / caso de uso: Unirse a una partida
Descripción: Reconectarse a una partida que anteriormente se había salido.	
Justificación: El sistema permitirá al usuario, que se haya desconectado de un partida, volver a conectarse. El sistema permitirá esta acción al usuario siempre que la partida no haya vuelto a continuar la partida. En el caso que no hayan más jugadores desconectados, la partida volverá a continuar en el mismo punto donde se había dejado.	
Dependencia: RF6, RF8	

Requisito: RF8	**Suceso / caso de uso:** Jugar
Descripción: Detener espera de reconexión del jugador desconectado.	
Justificación: El sistema debe permitir al usuario volver a continuar con la partida en caso que el jugador que se ha desconectado no tiene la intención de volver a reconectarse al juego. La partida volverá continuar en el mismo punto donde se había dejado, pero sin el jugador que se había desconectado. Solo en casos especiales, la mano que se estaba jugando será desechada y obligando a iniciar una nueva mano.	
Dependencia: RF6	

Requisito: RF9	**Suceso / caso de uso:** Ayuda
Descripción: Acceder a información relacionado con el juego y el sistema.	
Justificación: El sistema permitirá al usuario acceder a la información sobre las reglas del póker Texas Hold'em y sobre el funcionamiento del sistema.	
Dependencia: ---	

Requisito: RF10	**Suceso / caso de uso:** Salir y Jugar
Descripción: Salir de la partida.	
Justificación: El sistema debe permitir al usuario poder salir del juego.	
Dependencia: RF1, RF3, RF5	

Requisitos no funcionales

Conectividad

Requisito: RNF1	**Suceso / caso de uso:** ---
Descripción: Capacidad del producto software para restablecer conexión con otros dispositivos con el mismo producto de software.	
Justificación: Se desea que el juego pueda conectarse con otros dispositivos que tenga la misma aplicación para que puedan jugar todos juntos.	
Dependencia: ---	

Recuperabilidad (Fiabilidad)

Requisito: RNF2	Suceso / caso de uso: Unirse a una partida
Descripción: Capacidad del producto software para restablecer un nivel de prestaciones-especificado y de recuperar los datos directamente afectados en caso de fallo.	
Justificación: Se desea que el juego siempre que tenga una desconexión de un jugador se pueda recuperar y volver a continuar la partida.	
Dependencia: ---	

Mantenimiento

Requisito: RNF3	Suceso / caso de uso: Ayuda
Descripción: La aplicación debe ofrecer en la mayoría de pantallas un botón que dé acceso a la ayuda.	
Justificación: La aplicación debe dar información del funcionamiento de cada pantalla del juego y acceso a las reglas del juego.	
Dependencia: ---	

Usabilidad

Requisito: RNF4	Suceso / caso de uso: ---
Descripción: El sistema debe aprovechar las capacidades táctiles de la pantalla.	
Justificación: Se debe de aprovechar los diferentes gestos con los dedos disponibles para realizar acciones del juego de póker.	
Dependencia: ---	

5.3.3. Especificación – casos de uso

En el siguiente diagrama de casos de uso (Imagen 5.1) se describe la iteración de los seis casos que forma el sistema descrito anteriormente con un actor.

El actor que interviene es el usuario de la aplicación.

diagrama de los casos de uso

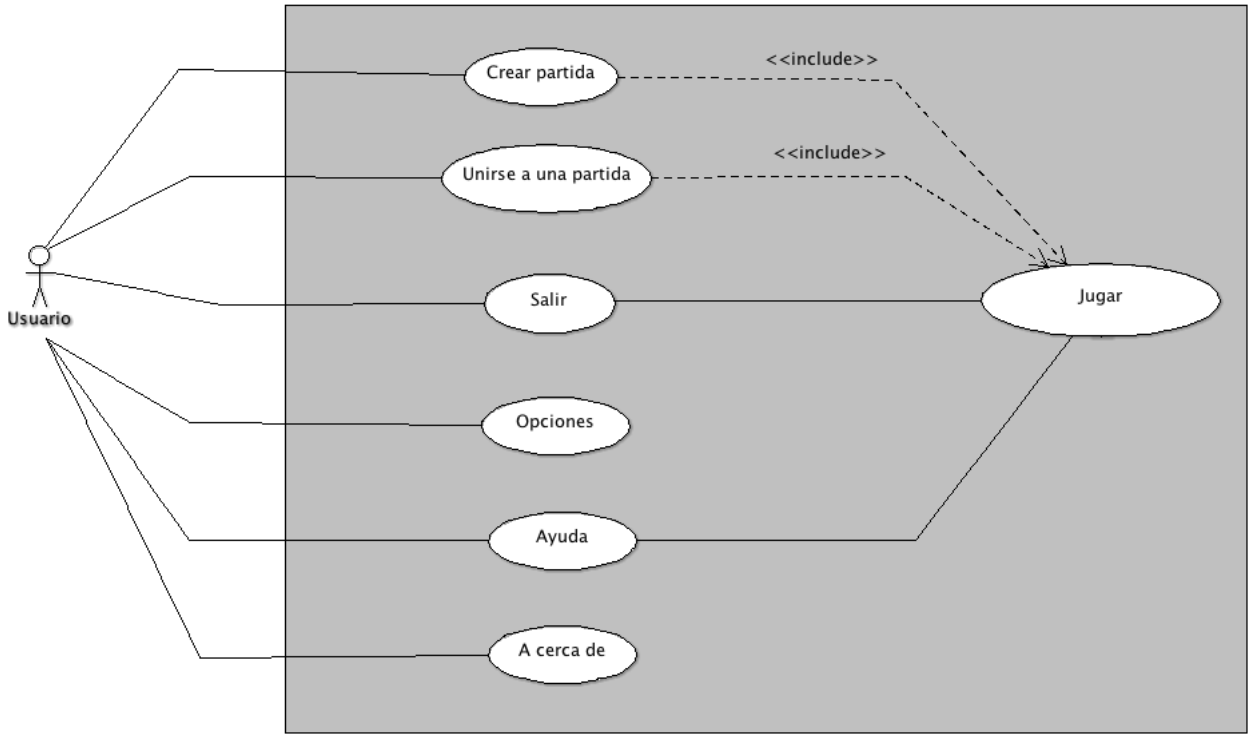

Imagen 5.1

descripción de los casos de uso

Caso de uso: Crear partida
Descripción: Creación de una nueva partida a partir de unos parámetros.
Actores: Usuario
Precondiciones: -
Inclusión de otros casos de uso: Jugar
Dialogo típico: El usuario configurará los diferentes parámetros de la partida como puntuación inicial, apuesta inicial, etc. Y una vez configurada, tendrá la partida creada y estará a la espera de la incorporación de nuevos jugadores.

Caso de uso: Unirse a una partida
Descripción: Unirse a una partida que ya sido creada y configurada.
Actores: Usuario
Precondiciones: Solo puede unirse a una partida si previamente alguien a creado una y si está no ha sido ya iniciada.
Inclusión de otros casos de uso: Jugar
Dialogo típico: El usuario buscará si hay disponible alguna partida ya creada y se unirá a ella. Una vez unido podrá saber que otros jugadores están ya unidos a la partida.

Caso de uso: Jugar
Descripción: Iniciar y jugar el juego póker Texas Hold'em.
Actores: Usuario
Precondiciones: Se debe haber creado previamente una partida (con su configuración) y haber unido como mínimo un jugador (siendo en total dos jugadores) para poder iniciar la partida.
Inclusión de otros casos de uso: -
Dialogo típico: El usuario que creó la partida espera el momento que todos los jugadores se hayan unido y inicia la partida. Una vez iniciada la partida, todos los jugadores podrán jugar al juego póker Texas Hold'em.

Caso de uso: Salir
Descripción: Salir de la partida.
Actores: Usuario
Precondiciones: Se debe de estar conectado a una partida.
Inclusión de otros casos de uso: -
Dialogo típico: El usuario decide salir de la partida actual produciendo la interrupción de ésta.

Caso de uso: Opciones
Descripción: ajuste de diferentes parámetros generales de la aplicación
Actores: Usuario
Precondiciones: -
Inclusión de otros casos de uso: -
Dialogo típico: El usuario puede ajustar parámetro generales de aplicación como, por ejemplo, el nombre del jugador, música o efectos especiales.

Caso de uso: Ayuda
Descripción: Información de ayuda de la aplicación y reglas del juego.
Actores: Usuario
Precondiciones: Solo se visualizará en la pantallas que sean necesarias.
Inclusión de otros casos de uso: -
Dialogo típico: El usuario podrá acceder información de ayuda de las diferentes pantallas y a las reglas del juego.

Caso de uso: A cerca de
Descripción: Descripción de información general sobre la aplicación.
Actores: Usuario
Precondiciones: -
Inclusión de otros casos de uso: -
Dialogo típico: El usuario podrá acceder a información relacionada con la aplicación como, por ejemplo, el autor y director de la aplicación.

5.3.4. Especificación – diagrama conceptual de los datos

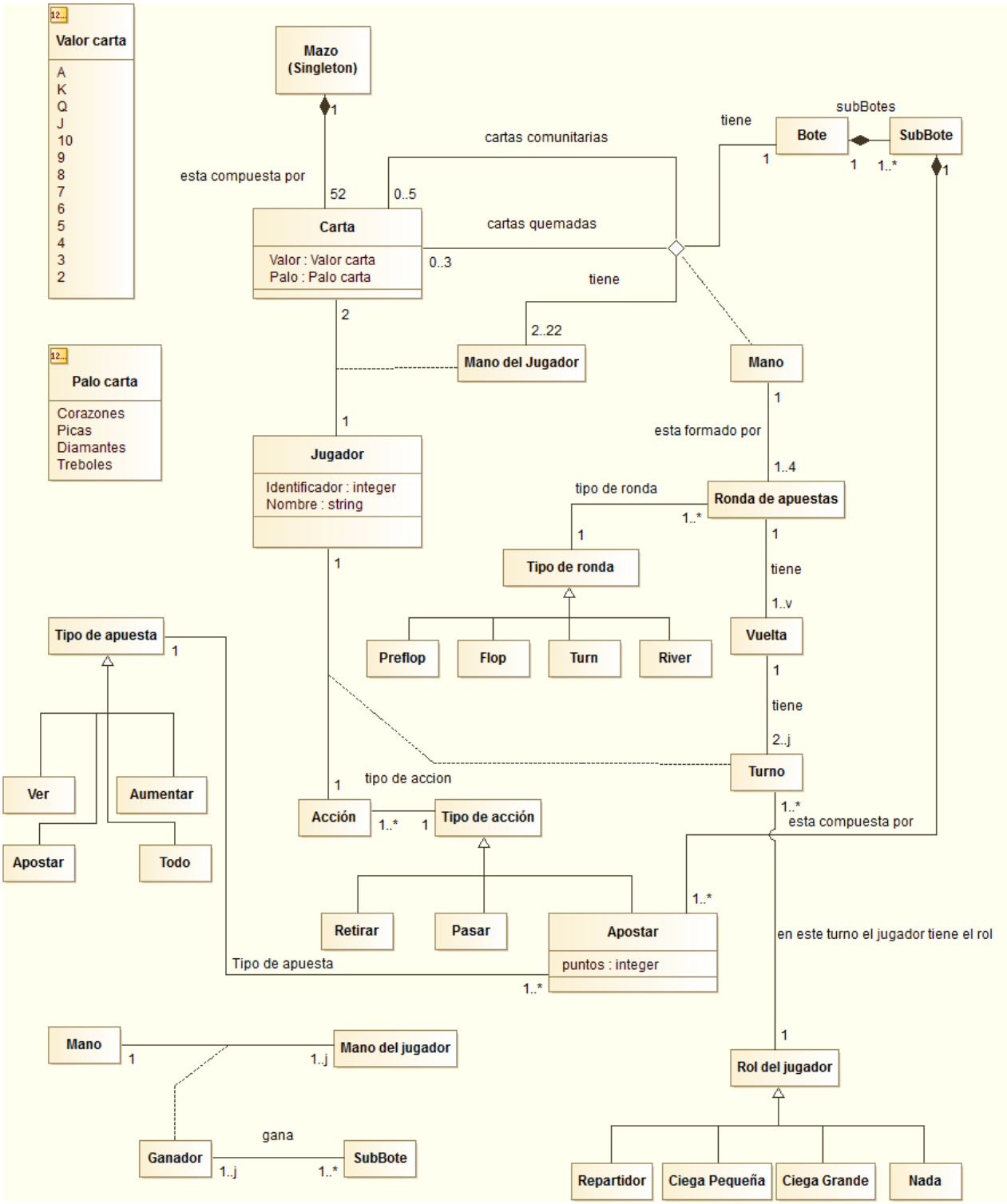

Imagen 5.2

Restricciones textuales:

- Llave externa: (Carta:Valor carta, Palo carta), (Jugador: Identificador)
- j = número de manos de jugador. v = número de vueltas necesarias para igualar todas las apuestas.

- La creación de instancias de “Ronda de apuestas” relacionadas a una instancia de “Mano” debe seguir el siguiente orden: Preflop, Flop, Turn y River.
- Si en una “Vuelta” la “Acción” decida por un “Jugador” es “Retirar”, el jugador no tendrá más “Turno”s en esa “Mano”.
- Un “Jugador” no puede “Apostar” más puntos de los que tiene.
- Cuando un “Jugador” apuesta (“Apostar”) todos los puntos que tiene, el “Tipo de apuesta” es “Todo”. Cuando pasa esto, el “Jugador” no obtendrá más “Turno”s en esa “Mano”, la “Acción” se tendrá en cuenta hasta el final de la “Mano” y se creará un nuevo “SubBote” si hay más “Jugador”es con opción de “Apostar”. El nuevo “SubBote” no podrá ser ganado por el “Jugador del “Todo”.
- Si al final de una “Vuelta” solo hay un jugador sin “Retirar”, este será el “Jugador” “Ganador” de uno o varios “Subbote”s.
- La “Mano de un jugador” que es ganador de una “Mano” debe pertenecer a esa “Mano”.
- Si se iguala todas las apuestas (“Apostar”) de una “Vuelta” de la “Ronda de apuestas” con fase “River” se asignará como uno o varios “Ganador”es, de su correspondiente “SubBote”, aquel que tenga la mejor combinación de 5 cartas correspondiente entre “Mano del jugador” y “cartas comunitarias”.
- Si un jugador se queda sin puntos este está fuera de juego y, por lo tanto, no podrá obtener más “Manos del jugador”.
- El ganador del juego será aquel que haya conseguido todos los puntos de los demás jugadores.

5.4. Diseño

En el apartado de diseño se verá unas pequeñas pincelas de diversos aspectos del diseño del sistema sin entrar en tanto detalles de mecanismos (como por ejemplo los protocolos) como se hizo en el capítulo anterior.

Tal como pasó en el capítulo anterior, los diferentes aspectos tecnológicos ya fueron escogidos antes de iniciar el proyecto. De la decisión de desarrollar una aplicación para iPhone, se han derivado los otros aspectos como el lenguaje (Objective-c), la interfaz de desarrollo (Xcode) y los frameworks (Cocoa Touch y Foundation). Otro framework o librería que también fue elegida de antemano es la librería P2P, desarrollada en el capítulo anterior. El único framework que no ha sido impuesto de antemano y que se ha escogido en esta fase del desarrollo es Cocos2d. El framework Cocos2d será utilizado para desarrollarlo de las vistas de la aplicación.

Los detalles referentes al diseño de las vistas de la aplicación se verán en el siguiente capítulo.

5.4.1. Utilización de la librería de P2P y su filosofía

Tal como se indico en el apartado 4.9 del anterior capítulo, la utilización de la librería está condicionada a una idea de utilización y ésta es que cada dispositivo que forma la red de la aplicación debe tener la misma información y la misma lógica de negocio. Siguiendo esta filosofía siempre que se desconecte alguien no habrá el problema de perdida de información.

Por lo tanto, la aplicación del juego de póker se tendrá que diseñar para que siempre comparta su información cuando se le conecte un nuevo jugador (o se reconecte).

Otro detalle sobre la utilización de la librería es la comunicación del objeto P2P con los demás objetos de la aplicación. Tal como se describirá más adelante, se utilizará la delegación para la comunicación entre objetos. En este caso hay varios punteros de delegación que se debe utilizar, todos menos uno son usados directamente sobre las vistas de conexión. El puntero delegado *p2pdelegate* es el único que no esta pensado para mostrar información por pantalla, sino que su misión es informar sobre otros eventos de conexión que los objetos principales de la aplicación, como “mainApp” y “Juego”, deben saber para su correcto funcionamiento.

5.4.2. Arquitectura en capas

Al igual que se hizo en el capítulo anterior, la arquitectura del software que se va seguir es la arquitectura en capas.

En el capítulo 3, apartado 3.3., se explicó que la arquitectura de una aplicación, que se obliga a seguir, es el patrón modelo vista controlador. Este patrón es un derivado de la arquitectura en capas con la utilización de 3 capas.

Siguiendo este concepto, se ha diseñado la aplicación para seguir las mismas capas obligatorias (modelo, vista y controlador) más una, comunicación. Esta cuarta capa será la encargada de realizar todo el trabajo de comunicación entre dispositivos que necesita la aplicación para poder jugar a póker. Todo el trabajo de la capa de comunicación está concentrada en la librería P2P y en un solo objeto, el P2P, y que se diseño en el capítulo anterior.

5.4.3. Diagrama de clases

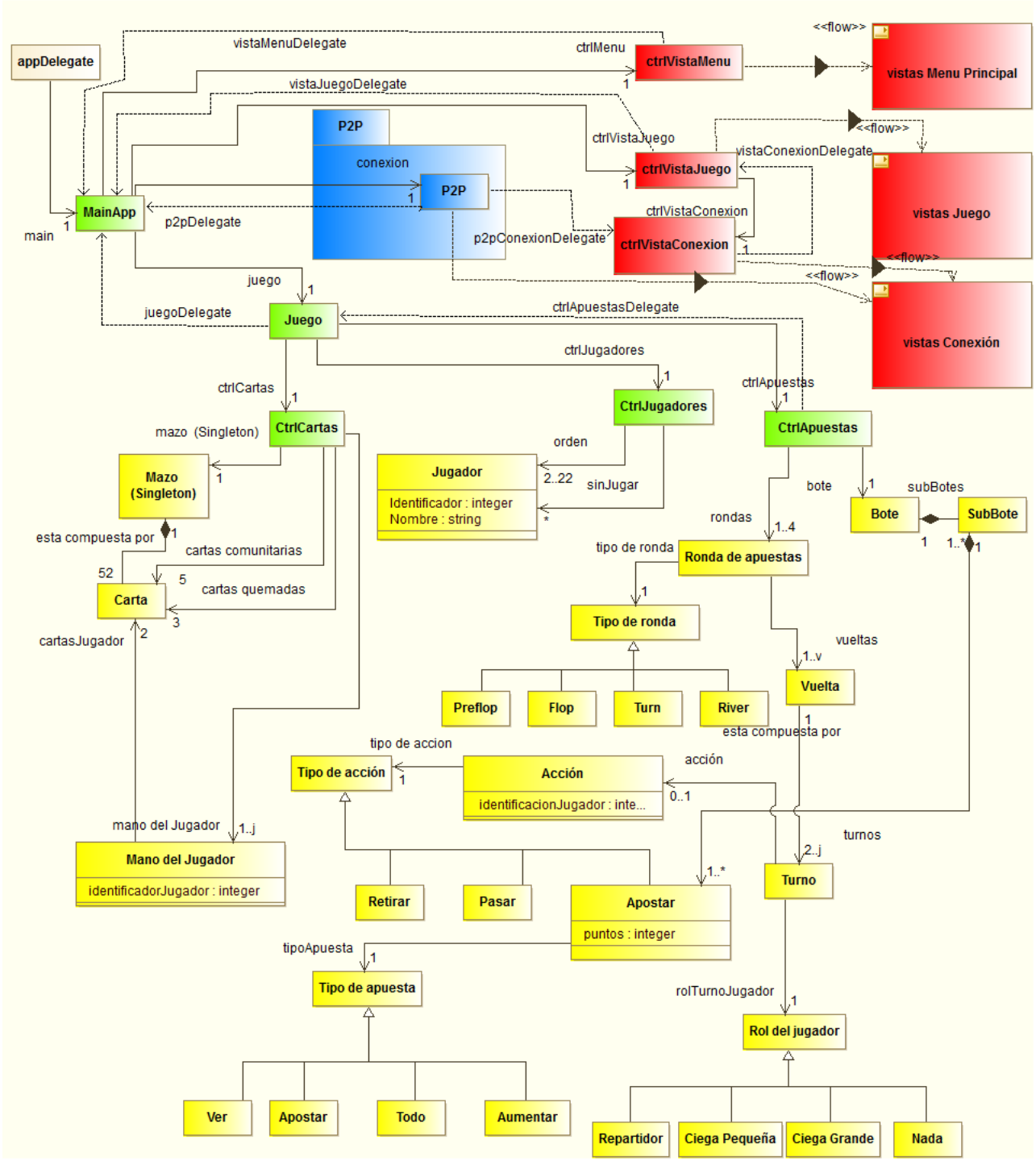

Imagen 5.3

En la imagen 5.3 se puede ver el diagrama de todas las clases que interviene en la aplicación. Las clases están coloreadas según la capa que pertenezca. Las clases de **color amarillo** son de la **capa de modelo**. Las clases de **color verde** son las de la **capa de controlador**. La clase y librería de **color azul** son de la **capa de comunicación**. Y las clases de **color rojo** son de la **capa de vista**.

En la capa de modelo se puede ver casi todas las clases dibujadas en la imagen 5.2 del apartado 5.3.4. Naturalmente los diagramas son diferentes ya que el diagrama de la imagen 5.2

representa la relación conceptual entre datos y la imagen 5.3 son objetos con relaciones simple que no necesariamente son los mismo que el otro diagrama. A pesar de las diferencias, el concepto es más o menos el mismo, representar diferentes entidades del sistema.

La capa de controlador esta formada por cinco clases. Las clases ctrlCartas, ctrlJugadores y ctrlApuestas son controladores frontera encargados de gestionar varias clases de la capa de dominio y ejecutar las lógicas de negocio relacionas con ellas. Estas tres clases anteriores son gestionadas por la clase Juego. La clase Juego controla la lógica general del juego de póker con ayuda de la información obtenida de los controladores frontera. La quinta y ultima clase de la capa de controlador es la MainApp. La clase MainApp gestiona la lógica general de la aplicación y enlaza la información de esta capa con las otras dos capas que a continuación se van a describir.

La capa de comunicación está formada únicamente por la clase P2P. Esta clase, que está enlaza con la librería P2P, se encarga de realizar las comunicaciones con otros dispositivos que necesita la clase Juego para utilizar la lógica del juego de póker.

La capa de vista esta formada por tres clases-controladores que controla los tres tipos de vista que hay en la aplicación, menú, juego y conexión, y por varias clases que son las diferentes pantallas de la aplicación que están controladas por estas tres primeras. A pesar que ctrlVistaMenu, ctrlVistaJuego y ctrlVistaConexion son controladores, éstos han sido incluidos en la capa de vista porque su papel no tiene mucha implicación con la lógica del juego de póker. Las demás clases de vista serán detalladas en el siguiente capítulo.

La única clase que no ha sido alineada a ninguna de las capas es appDelegate, que tal como se describió en el capítulo 3, apartado 3.3.2., tiene la función de realizar las primeras tareas de inicialización de la aplicación y la función de recibir diferentes eventos generales de la aplicación.

5.4.4. Patrón delegación

Otra característica muy utilizada en el diseño de la aplicación es la utilización de varios patrones de diseño, entre ellos el patrón delegación. El patrón delegación consiste en ceder a una clase a otra la responsabilidad de implementar ciertas funcionalidades. Con la delegación de responsabilidades, la primera puede llamar a la segunda para obtener el resultado de una funcionalidad que originalmente era responsabilidad suya.

Este patrón es muy utilizado en el iOS SDK para que el desarrollador personalice o extienda el comportamiento de algunas clase y/o comunicar ciertos eventos entre clases. El Objective-c necesita utilizar el mecanismo de protocolo, explicado en el capítulo 3, apartado 3.1.5., para poder implementar este patrón. Una gran ventaja que da este patrón es que permite reusar y/o remplazar las clases delegadas que implementa los métodos delegados (definidos en un protocolo) por otra que tenga los mismo métodos delegados implementados. Con esta propiedad, se baja el

acoplamiento entra la clase original y la clase delegada porque la clase original no estará relacionada fuertemente con la clase delegada.

En el diagrama de clases de la imagen 5.3, se puede ver varias clases utilizando delegados para comunicarse con otras o para delegar responsabilidades (son aquellas que tiene una flecha discontinua). Un ejemplo muy claro es la librería P2P que tiene el delegado p2pdelegate para comunicarse con la clase mainApp y tiene otros delegados para delegar la responsabilidad de mostrar por pantalla el estado de la conexión.

5.4.5. Patrón singleton

El patrón singleton consiste en prohibir a una clase que pueda tener más de una instancia o objeto en el sistema. Al limitar la clase a una instancia, el sistema tendrá un mayor control y proporcionará un punto global sobre ésta.

En el diagrama de clases de la imagen 5.3, se puede ver indicado que la clase Mazo es una clase singleton. La razón para convertir esta clase en singleton es la necesidad de controlar la creación de instancias de la clase Carta. Si no estuviera Mazo o hubiera varias instancias de Mazo, la creación de las instancias de carta no estaría controlada, dando la posibilidad de crear, sin querer, dos cartas con el mismo palo y valor.

5.4.6. Seguridad

Tal como se aviso en el ultimo apartado del capítulo anterior, la responsabilidad de implementar la seguridad en las comunicaciones es del desarrollador que utilice la librería P2P. En el caso de este desarrollo se ha decido no diseñar ningún proceso de seguridad porque en la especificación del sistema no lo ha indicado (normalmente estaría indicado en el apartado de requisitos no funcionales). La razón de la ausencia de ésta es porque no es un aspecto clave para un juego de póker.

Es verdad que sin seguridad se podría hacer trampas pero sería mucho trabajo para realizarlo en un simple juego y sería más o menos fácil de ver cuando un jugador o varios están realizando trampas.

Si la aplicación hubiera sido una aplicación empresarial que compartiera información confidencial entre varios dispositivos, la necesidad de diseñar un proceso de seguridad hubiera sido evidente y explicita en la especificación del sistema.

5.5. Codificación – implementación

Para la implementación de la aplicación se ha creado un nuevo proyecto en el Xcode utilizando la plantilla “view-base application”. Una vez creado el proyecto, se unió dos librerías necesarias para el proyecto, la librería P2P desarrollada anteriormente (junto con la declaración de la clase P2P) y la librería del Cocos2d. Para la utilización de la librería Cocos2d se tuvo que realizar cambios y añadir código muy especifico en las clases appDelegate, mainApp y en varios ctrlVista.

Una vez preparado el proyecto en el Xcode con las librerías necesarias para la implementación, se inició con normalidad la implementación del resto de la aplicación.

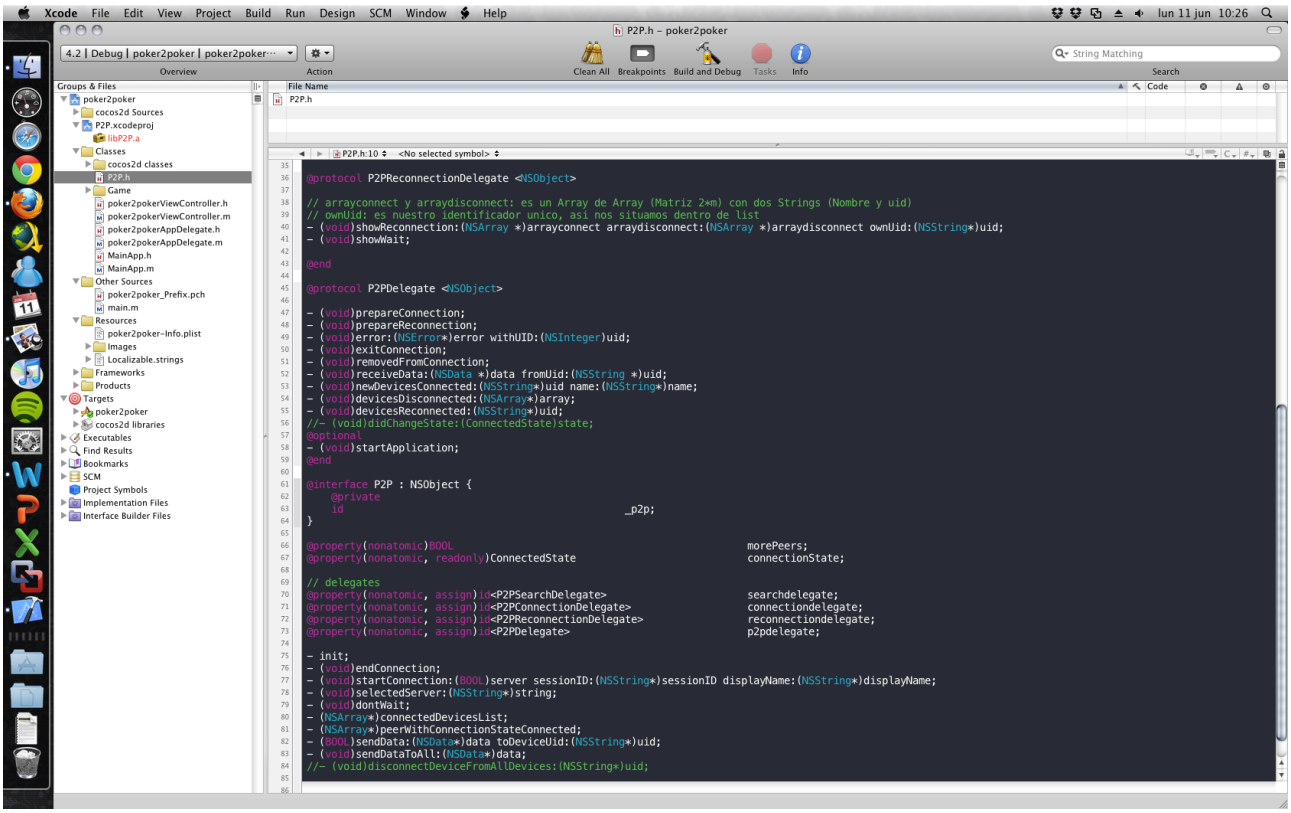

Imagen 5.4

5.6. Pruebas

Las pruebas del correcto funcionamiento de la aplicación se fueron realizando a medida (de forma incremental) que se iban implementado las diferentes partes de la aplicación.

La implementación comenzó con la creación de las clases de controladores frontera que controlan los tres aspecto fundamentales del juego de póker: cartas, jugadores y apuestas.

Una vez creada la implementación del ctrlCartas, con sus correspondientes clases de la capa de modelo, se realizó las pruebas unitarias de la clase que son: generación aleatoria de cartas únicas repartidas entre jugadores, de las cartas quemadas y de las cartas comunitarias; la identificación de la mejor jugada que tiene cada jugador y identificación del jugador o jugadores ganadores. Naturalmente, las pruebas unitarias buscaban la detección de posibles fallos en los casos extremos

como, por ejemplo, encontrar la mejor jugada cuando hay varias posibles muy parecidas o cuando los jugadores tiene la misma jugada y hay que detectar si es empate o no.

De la misma manera también se procedió a la pruebas unitarias de ctrlJugadores y ctrlApuestas, implementando sus respectivas clases de la capa de modelo primero y realizando las pruebas que detectarán los fallos en los casos extremos.

En los tres controladores, las pruebas unitarias se realizaron con datos ficticios de los demás controladores. Pero estos ya fueron los reales cuando se implementó la clase Juego, que permitió la comunicación entre los tres. Además de estas pruebas, la clase Juego también pudo realizar las pruebas de funcionamiento general de la lógica del juego de póker utilizando los datos de los tres controladores frontera.

De esta forma gradual se fue implementado y realizando las pruebas con el mainApp, con las comunicaciones de la clase P2P y con las clases de la capa de vista.

A pesar que las clases de la vista se quedaron para el final de la implementación, todas las pruebas de las primeras clases implementadas se hicieron con la utilización de vistas sencillas (creadas solo para ese propósito) que simplemente servían para comprobar el correcto funcionamiento de las pruebas.

6. COCOS2D – LA INTERFAZ DE USUARIO

Éste es el último capítulo de desarrollo de este documento y, junto con el capítulo anterior, abarca el segundo objetivo del proyecto final de carrera. De los tres capítulos de desarrollo (Comunicación, Juego-Poker y Cocos2d-la interfaz de usuario), éste es el más corto y secundario.

El contenido de este capítulo ha sido separado del capítulo anterior porque era necesario introducir nuevos conceptos relacionados con la librería Cocos2d. Aun así, hay que volver a recordar que el contenido de este capítulo y el anterior hacen una unidad que es el desarrollo de la aplicación del juego de póker Texas Hold'em.

6.1. Introducción

En el capítulo anterior se realizó todo el proceso de desarrollo de la aplicación del juego de cartas, pero en el apartado de diseño, apartado 5.4., se omitió un aspecto importante, la interfaz de usuario.

Normalmente, para la interfaz de usuario se recorrería a utilizar los frameworks o librerías de la capa de Cocoa Touch, en concreto el UIKit que contiene las vistas y controles que normalmente se utilizan en las aplicaciones de iOS. El problema está en que se quiere diseñar una interfaz con muchos gráficos e imágenes relacionadas con el póker y que, además, puedan interactuar con los diferentes gestos del usuario. Pero las vistas y los controles por defecto del UIKit no son adecuados para este fin. Por lo tanto, es necesario desarrollar una interfaz personalizada para el juego.

El iOS SDK[15] ofrece dos alternativas para dibujar gráficos: la librería Quartz 2d, que pertenece al framework Core Graphics, y la librería OpenGL ES, la librería de gráficos multisistema más famosa del mundo. Ambas alternativas ofrecen muchas clases y métodos que cumpliría perfectamente con el objetivo, pero la utilización y el aprendizaje de ambas librerías son complejas.

Durante la búsqueda de generación de gráficos en el iOS se encontró una tercera alternativa, la librería Cocos2d. Esta librería, en comparación con las otras dos, es bastante más fácil de utilizar y permite realizar bastantes funcionalidades que con las otras dos hubieran sido necesarios la utilización de bastantes líneas de código. Por esta razón, se ha decidido utilizar esta librería para el desarrollo de la interfaz de usuario.

[15] Software Development Kit o, traducido al castellano, kit de desarrollo de software.

6.2. Cocos2d

6.2.1. Introducción

Cocos2d es una librería que no pertenece al conjunto de frameworks del iOS SDK. Esta librería fue creada por la comunidad *open source* para la generación de gráficos 2d en aplicaciones, como por ejemplo los videojuegos. La utilización del Cocos2d en el desarrollo de aplicaciones de iOS es aceptada por Apple.

La librería está implementada con el lenguaje Objective-c y el OpenGL ES. La integración de la librería en la arquitectura de una aplicación de iOS se realiza mediante la utilización del UIViewController principal y una subclase del UIView que trabaja específicamente para OpenGL ES. Utilizando únicamente esta vista, el Cocos2d muestra todos los gráficos que generan los diferentes componentes que integra su librería.

En los siguientes apartadas se describen los principales componentes de cocos2d:

6.2.2. CCDirector

La clase CCDirector o director es el corazón del motor gráfico de Cocos2d. Esta clase utiliza el patrón sinlgeton[16] para ser accesible desde cualquier punto del código porque la función de esta clase es la de almacenar la configuración global y gestionar las escenas de Cocos2d.

Fundamentalmente el uso que se le da al director es:

- Acceder a la escena actual y cambiar de escena.
- Acceder a la configuración de cocos2d.
- Pausar, retomar y acabar el funcionamiento general de todos los componentes de cocos2d.

A continuación, un ejemplo de inicio de la primera escena de la aplicación:

```
SceneMenuPrincipal *mainScene = [SceneMenuPrincipal node];
[[CCDirector sharedDirector] runWithScene: mainScene];
```

Una característica importante de la gestión de escenas es que el director solo puede tener en ejecución una escena. Si el desarrollador quiere utilizar otra escena deberá remplazar la actual por la otra.

Desde una escena se inicia una jerarquía de componentes o nodos, en forma de grafo, que todos juntos constituye la representación visual de toda la escena. Cada nodo solo puede tener un padre y tiene varios hijos (una escena no tiene padre).

[16] Visto en el capítulo 5, apartado 5.4.5.

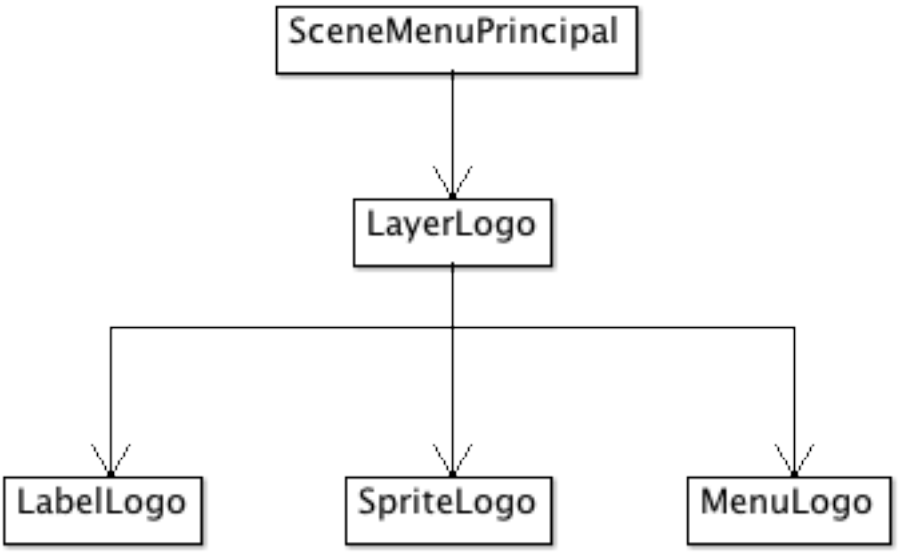

Imagen 6.1

6.2.3. CCNode

La clase CCNode o nodo es la clase base desde donde heredan la gran parte de los componentes de Cocos2d, clases como CCScene, CCLayer, CCSprite o el CCLabel son un buen ejemplo. Esta clase en si misma no tiene ninguna representación visual, simplemente define unas propiedades y unos métodos que deben tener todos nodos. Las características más importantes que ofrece un nodo son:

- Gestión de la jerarquía del grafo de nodos. Funcionalidades como añadir, recuperar o borrar nodos hijos desde un nodo padre.
- Gestión de acciones (clase CCAction) como por ejemplo mover un nodo de un lugar a otro en un tiempo determinado, rotar un nodo en unos grados y en un tiempo determinado, etc.
- Programación de llamadas automáticas en intervalos de tiempo regulables.

A continuación, un ejemplo de recuperación de un nodo (un CCLayer) y la realización de una acción para moverlo:

```
CCLayerColor *logo = (CCLayerColor*)[self getChildByTag:LOGO];
CCMoveTo* move = [CCMoveTo actionWithDuration:2 position:CGPointMake(-90,0)];
[logo runAction:move];
```

6.2.4. CCScene

La clase CCScene o escena, ya vista en el apartado 6.2.2., es simplemente el primer nodo del grafo de nodos. Al igual que le pasan a los nodos, la clase CCScene no tiene representación visual por si sola. Prácticamente, una escena no tiene más funciones.

Normalmente, el primer nivel jerárquico de los hijos de una escena son los layers (CCLayer).

A continuación, un ejemplo de añadir un layer a una escena y el cambió del escena actual por otra:

```
CCLayer *layerMenu = [LayerMenuPrincipal layerWithColor:ccc4(255,255,255,255)];
SceneMenuPrincipal *mainScene = [SceneMenuPrincipal node];
[mainScene addChild:layerMenu];
[[CCDirector sharedDirector] replaceScene:mainScene];
```

En el diseño de la capa de presentación, la clase escena sería una vista que a partir de ella se añadiría todos los componentes visuales.

6.2.5. CCLayer

La clase CCLayer o Layer es una clase, que siguiendo la jerarquía de nodos explicada anteriormente, tiene la misión de agrupar varios nodos (siendo hijos de éste). En cierta manera, esta clase recuerda un poco a la clase CCScene, pero en realidad la clase CCLayer tiene más características importantes:

- Una escena puede contener varios Layers. De esta manera, se puede dividir el trabajo de visualización en diferentes partes con acciones distintas.
- Recepción de eventos de gestos sobre la pantalla.
- Recepción de eventos del acelerómetro.

A continuación, un ejemplo de la declaración de los métodos que reciben los eventos de gestos en una clase que hereda de la clase CCLayer:

```
-(BOOL) ccTouchBegan:(UITouch *)touch withEvent:(UIEvent *)event;
-(void) ccTouchMoved:(UITouch *)touch withEvent:(UIEvent *)event;
-(void) ccTouchEnded:(UITouch *)touch withEvent:(UIEvent *)event;
```

6.2.6. CCSprite

La clase CCSprite o Sprite es la clase más utilizada de la librería Cocos2d. Está clase es la encargada de visualizar por pantalla las texturas creadas por código o las imágenes de un archivo. A partir de aquí, como esta clase hereda de la clase CCNode, el objeto de tipo CCSprite podrá moverse, escalar su imagen, etc.

A continuación, un ejemplo de un sprite que carga una imagen desde un archivo:

```
CCSprite* sprite = [CCSprite spriteWithFile:@”Default.png”];
 [self addChild:sprite];
```

6.2.7. CCLabel

La clase CCLabel es una clase parecida al sprite pero con la diferencia que en vez de visualizar una imagen, visualiza un texto. Naturalmente, esta clase permite configurar propiedades normales de un texto como es la fuente, el tamaño o el color.

A continuación, un ejemplo de creación de un Label:

```
CCLabel* label = [CCLabel labelWithString:@"texto" fontName:@"AppleGothic"
fontSize:32];
[self addChild:label];
```

6.2.8. CCMenu

La clase CCMenu es la clase que proporciona la creación de menús con sus respectivos botones (CCMenuItem). Los botones de estos menús pueden ser un texto (como si fuera un CCLabel) o una imagen (como si fuera un CCSprite). Otra característica que tiene estos botones es que cuando un botón es apretado, realiza la llamada a un método que se cargará de gestionar la lógica del botón.

A continuación, un ejemplo de creación de un Menú:

```
[CCMenuItemFont setFontName:@"Helvetica-BoldOblique"];
[CCMenuItemFont setFontSize:26];

CCMenuItemFont* boton1 = [CCMenuItemFont itemFromString:@"Boton1"
target:selfselector:@selector(menuItem1Touched:)];

CCSprite* normal = [CCSprite spriteWithFile:@"Icono.png"];
normal.color = ccRED;

CCSprite* seleccionado = [CCSprite spriteWithFile:@"Icono.png"];
seleccionado.color = ccGREEN;

CCMenuItemSprite* boton2 = [CCMenuItemSprite itemFromNormalSprite:normal
selectedSprite:seleccionado target:self selector:@selector(menuItem2Touched:)];

CCMenu* menu = [CCMenu menuWithItems:boton1, boton2, nil];
menu.position = CGPointMake(size.width / 2, size.height / 2);
[self addChild:menu];
```

6.3. Diseño

6.3.1. Distribución de las pantallas

La primera tarea en el diseño de una interfaz de usuario es saber que pantallas serán necesarias a diseñar. El conjunto de pantallas a diseñar debe completar todas la interacciones posibles que hayan entre el usuario y el sistema (la aplicación).

Si se visualiza la imagen 5.3, del apartado 5.4.3. del capítulo anterior, se puede ver que las vistas están englobadas en tres grupos diferentes. Estos tres grupos son vistas del menú principal, vistas de conexión y vistas del juego.

Las vistas del menú principal se basan más o menos de los diferentes casos de uso que se vieron en el capítulo 5. del capítulo anterior. Los casos de uso que no tienen relación directa con el juego son aquellos que deben ser accesibles desde de la primera pantalla (la pantalla principal). Por lo tanto, además de la vista de menú principal, también se consideran del grupo de menú las siguientes vistas: la vista de configuración de una nueva partida, la vista de búsqueda de una partida (esta también pertenece a las vistas de conexión), la vista de opciones y la vista de acerca de.

Las vistas de conexión son aquellas que son necesarias para la creación, la búsqueda y el mantenimiento de la red en que funcionará el juego. Las vistas de conexión se basan directamente de los delegados de tipo vista de la librería P2P, definidos en el diagrama del apartado 4.6.10 del capítulo 4.. Las vistas que corresponden a estos tres delegados son la de búsqueda, la de dispositivos conectados y la de reconexión de dispositivos.

Las vistas del juego son aquellas que ofrecen información del estado del juego y aquellas que permiten realizar las diferentes acciones de un jugador. Las vistas que ofrecerán información del juego son tres: la vista principal de juego que muestra el estado actual, la vista de cambio de estado de las cartas comunitarias y la vista de finalización de mano. Las vistas que permiten realizar acciones del juego son dos: la vista de turno y la vista de apostar (en el caso que en la vista de turno se haya decido apostar).

6.3.2. Mapa navegacional de las pantallas

A continuación se muestra el mapa navegacional de las pantallas que se han citado en el apartado anterior.

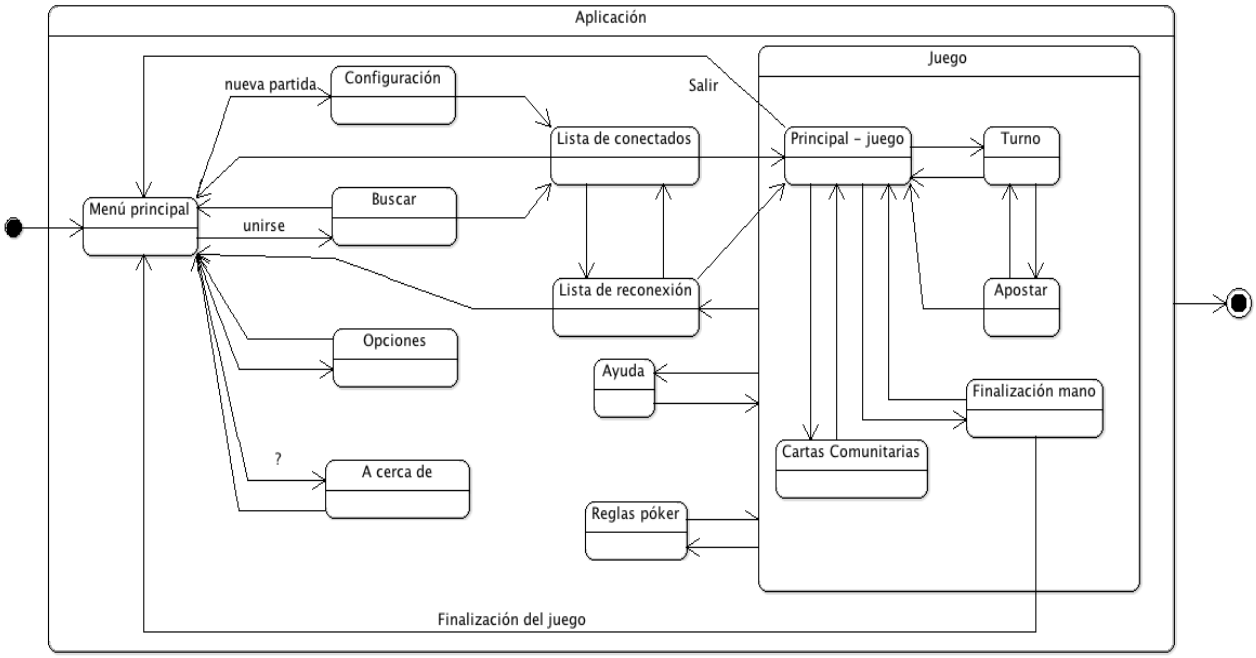

Imagen 6.2

La imagen 6.2 muestra las relaciones que existen entre las pantallas y el sentido en que se puede navegar de una a otra. El objetivo de este diagrama de estados es mostrar los diferentes caminos que el usuario puede moverse en la aplicación, partiendo desde el inicio de la pantalla principal hasta cualquier punto del diagrama en que el usuario decida salir de aplicación (apretando el botón físico principal del iPhone).

6.3.3. Criterios generales

Este apartado se describen cuales son los criterios comunes que deben seguir todas o la mayoría de pantallas de la aplicación. Estos criterios se dividen dos grupos, estéticos y de comportamiento:

Estéticos

Los criterios estéticos de las pantallas tienen el objetivo de marcar una serie de directrices que deben seguir todas para que su aspecto sea homogéneo en toda la aplicación. Las directrices son los diferentes aspectos que definen estéticamente una pantalla como, por ejemplo, colores, fuentes, formas de los diferentes controles, etc. Normalmente, estos aspectos suelen ser definidos por la empresa que contracta el desarrollo de una aplicación. En la mayoría de los casos, la empresa prefiere seguir el diseño de su logo corporativo.

Imagen 6.3

Siguiendo la idea de utilizar un logo como base del diseño, se ha escogido **el logotipo de la UPC** para el diseño de la aplicación. A partir de este logotipo, se han decidido las siguientes directrices para el diseño de las pantallas :

- **Colores**: Utilizar el mismo color azul del logo como color predominante, siendo utilizado en los textos, en los controles, en los gráficos y en las imágenes. El color blanco del fondo del logotipo también será el color del fondo de las pantallas. Si hay la necesidad de utilizar otros colores, se debe escoger el menor número posible de colores diferentes para no perder la homogeneidad del color azul.

- **Texto**: Se debe utilizar la misma fuente en todas pantallas y un rango de tamaños los más parecidos posibles.

- **Formas de los controles**: Siguiendo la forma redondeada del logo, los controles como los campos de texto, los campos numéricos, los botones, etc. deben ser de formas redondeas, evitando lo máximo posible las esquinas rectangulares.

- **Imágenes**: Siguiendo la misma idea del punto anterior, las imágenes deben tener acabados redondeados evitando lo máximo posible las esquinas rectangulares.

- **Distribución**: El logo es sencillo, con poca carga visual de elementos, haciéndola agradable visualmente. La distribución de las pantallas deben ser iguales que el logo, deben ser pantallas sencillas que eviten, lo máximo posible, la recarga de elementos visuales. En otras palabras, las pantallas deben seguir un diseño minimalista.

Comportamiento

Normalmente las pantallas se comportan de una manera muy pauta y rutinaria que les hacen ser fáciles de seguir por parte del usuario. Por ejemplo, si el usuario toca un botón, la lógica que hay detrás del botón realizará un cambio inmediato en la pantalla.

La aplicación que se está desarrollando no solo interviene los eventos provocados por el usuario, sino que también intervienen otros eventos realizados por otras entidades como son los demás jugadores del juego y la red.

Por lo tanto, en el grupo de vistas del juego, la lógica que gestiona los cambios de pantalla debe tener en cuenta que hay muchos cambios en las vistas, originados por los demás jugadores del juego y no por el usuario de la aplicación. Aun así, los eventos de los otros jugadores son previsibles y pautados porque siguen la dinámica del juego de cartas.

El pequeño problema que el controlador de las vistas del juego debe controlar es la interrupción de la partida a causa de la desconexión de un jugador. Esto es un pequeño problema porque la interrupción puede suceder en cualquier vista del juego. Así que cuando se realice el diseño de la lógica de gestión de pantallas se debe tener en cuenta este comportamiento.

6.4. Pantallas

En este apartado se repasan todas las vistas de la aplicación, mostrando una imagen de cada una de las pantallas y describiendo brevemente su funcionamiento.

Se debe recordar que cada una de estas pantallas son clases que heredan de la clase CCScene.

6.4.1. Menú principal

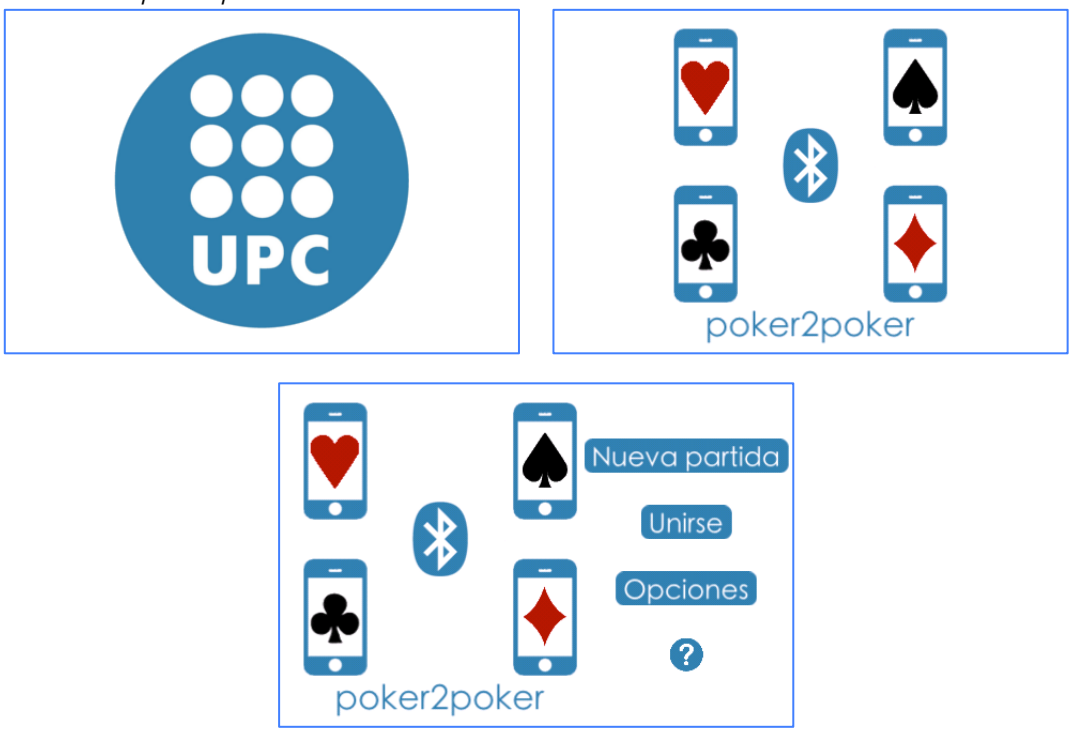

Imagen 6.4

Al iniciar la aplicación, la primera imagen que aparece es logo de la upc y, a los pocos segundos, aparece el logo y el título de la aplicación. Si el usuario toca la pantalla, el logo y el título se desplazan y dejan visualizar los cuatro botones del menú principal: Nueva partida, unirse, opciones y acerca de (?).

6.4.2. Aviso del Bluetooth

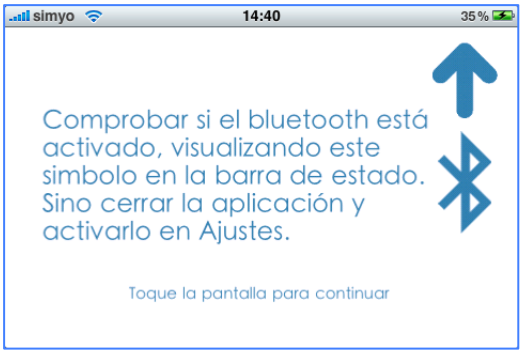

Imagen 6.5

Ésta es la primera pantalla que aparece una vez seleccionado el botón Nueva partida o el botón Unirse de la pantalla de menú principal. Esta pantalla es una pantalla previa antes de visualizar las pantallas de configuración de la partida (botón nueva partida) y de buscar partida (botón unirse). El objetivo de esta pantalla es informar al usuario que debe tener activado el Bluetooth para poder crear o unirse a una partida. Para ayudar en el proceso de comprobación, se muestra la barra de estado del iOS , donde debería aparecer el signo de Bluetooth.

6.4.3. Configuración de la partida

Imagen 6.6

La pantalla de configuración tiene el objetivo de permitir al usuario ajustar diferentes parámetros de la partida que está apunto de crear.

6.4.4. Buscar partida

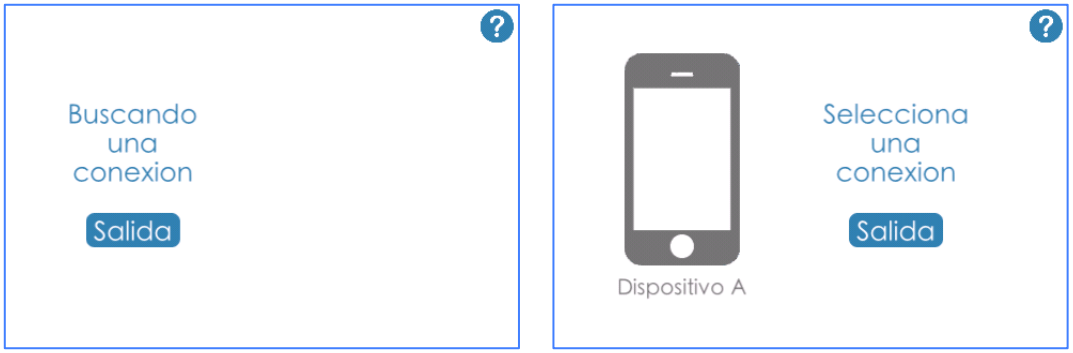

Imagen 6.7

La pantalla de búsqueda tiene el fin de ayudar al usuario a buscar una conexión de una partida y a conectarse a ella.

6.4.5. Opciones

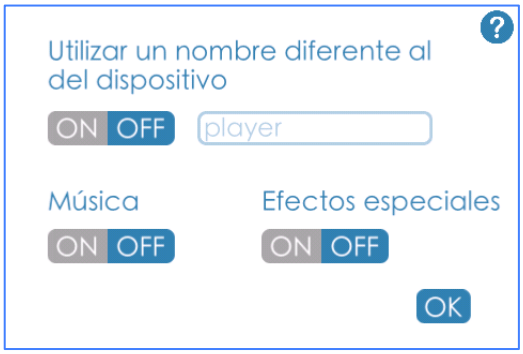

Imagen 6.8

La pantalla de opciones tiene el objetivo de permitir al usuario configurar varios aspectos generales del juego como, por ejemplo el nombre del jugador, la música o los efectos especiales.

6.4.6. A cerca de

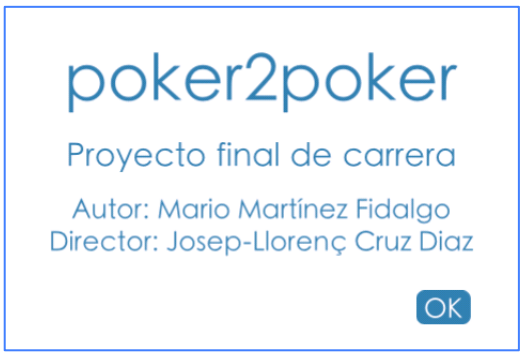

Imagen 6.9

Esta pantalla da información referente a los autores y otros datos de la aplicación.

6.4.7. Lista de conectados

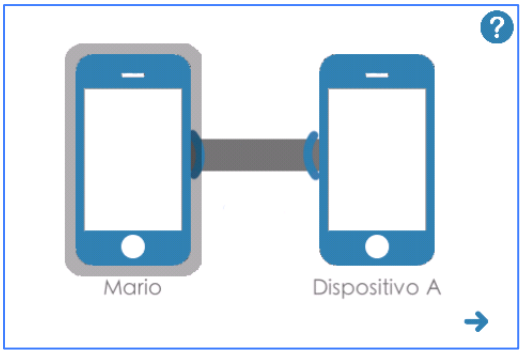

Imagen 6.10

La lista de conectados informa al usuario cuales son los dispositivos o los jugadores que están conectados a la partida antes que ésta se inicie. La primera imagen muestra una lista con solo un jugador conectado (el usuario que ha creado la partida). La segunda imagen muestra una lista de dos jugadores conectados. Y la tercera imagen muestra la misma pantalla anterior, pero desplazada hacia la derecha para poder mostrar los botones de inicio o salida de la partida (el botón de inició solo lo tiene el jugador que ha creado la partida).

6.4.8. Lista de reconexión

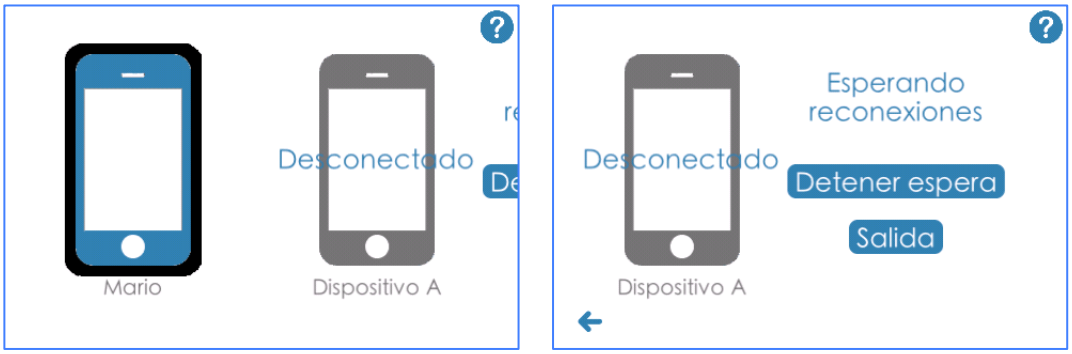

Imagen 6.11

La vista de lista de reconexión tiene el fin de informar al usuario que ha habido una desconexión de un jugador o un dispositivo. La información que muestra esta pantalla es la siguiente: los jugadores conectados y los jugadores desconectados a la red. A partir de esta pantalla, los jugadores conectados esperan la reconexión de los jugadores desconectados. En el caso de no tener reconexiones, el jugador creador de la partida dará al botón “Detener espera” para retomar la partida con los jugadores conectados.

El comportamiento de esta pantalla es la misma que la vista de lista de conectados, permite el desplazamiento lateral para visualizar todos los elementos de la vista.

6.4.9. Vista principal del juego

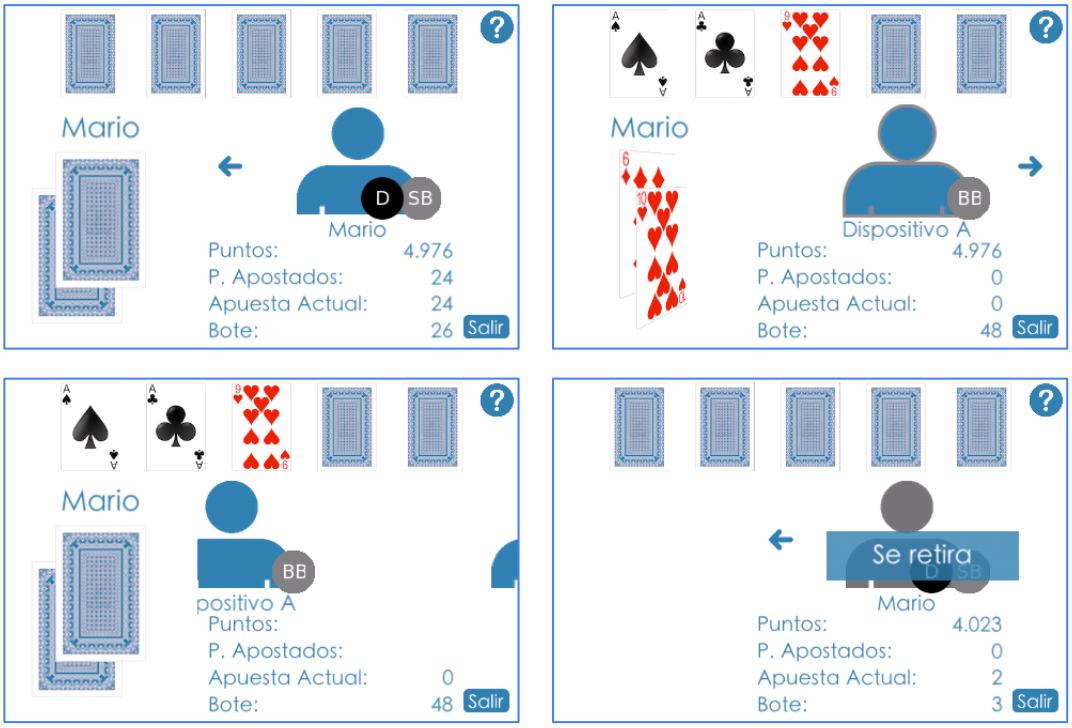

Imagen 6.12

La vista principal del juego informa al usuario de la situación actual de la partida de póker. La información que muestra esta vista es la siguiente:

- Las **cartas comunitarias**: según vaya pasando los turnos de las apuestas, se irán destapado una o varias cartas.
- Las **cartas del usuario**: mediante el gesto de desplazamiento lateral sobre las dos cartas de la izquierda se descubre el valor de éstas. Si el usuario no tiene cartas eso significa que se ha retirado o que está fuera de la partida.
- **Información de todos los jugadores**: mediante el gesto de desplazamiento lateral del muñeco del centro de la pantalla se irá accediendo a la información de cada jugador. Un muñeco representa un jugador y la información que aporta es la siguiente:
  - El nombre del jugador.
  - El estado que está el jugador: Si está en juego, si se ha retirado y su rol en esa mano.
  - El jugador del turno actual.
  - Los puntos que tiene el jugador.
  - Los puntos apostados por el jugador en esa ronda de apuestas.
- La **apuesta actual**: la apuesta más alta en la ronda actual de apuestas.
- El **bote**: la cantidad de puntos acumulados en apuestas en la mano actual.

6.4.10. Cartas comunitarias

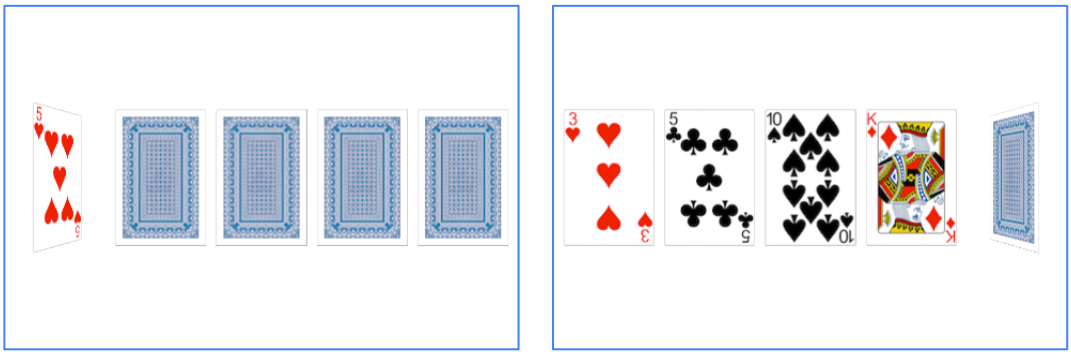

Imagen 6.13

La pantalla de cartas comunitarias es una animación que dura pocos segundos. La animación indica al usuario que se ha terminado una ronda de apuestas y revela el valor de una o varias cartas comunitarias.

6.4.11. Turno

Imagen 6.14

La vista de turno informa al usuario que el turno actual es suyo. Esta pantalla informa del estado actual del usuario y de los datos esenciales para tomar una decisión en su turno. Al igual que la vista principal del juego, el usuario podrá ver el valor de sus cartas mediante el gesto de desplazamiento lateral sobre las cartas. Al realizar el gesto de lateral de las cartas del usuario, aparecerán las cartas comunitarias actuales. Los diversos gestos que puede hacer el usuario para indicar su acción en ese turno son los siguientes:

- Dar **dos toques** sobre las cartas para indicar que **pasa**.
- **Desplazar** hacia **arriba** de la pantalla las cartas para indicar que **se retira**.
- **Desplazar la ficha** hacia las cartas para indicar que quiere **apostar**.

Al realizar la acción de apostar, se cambiará la vista de turno por la vista de apostar.

6.4.12. Apostar

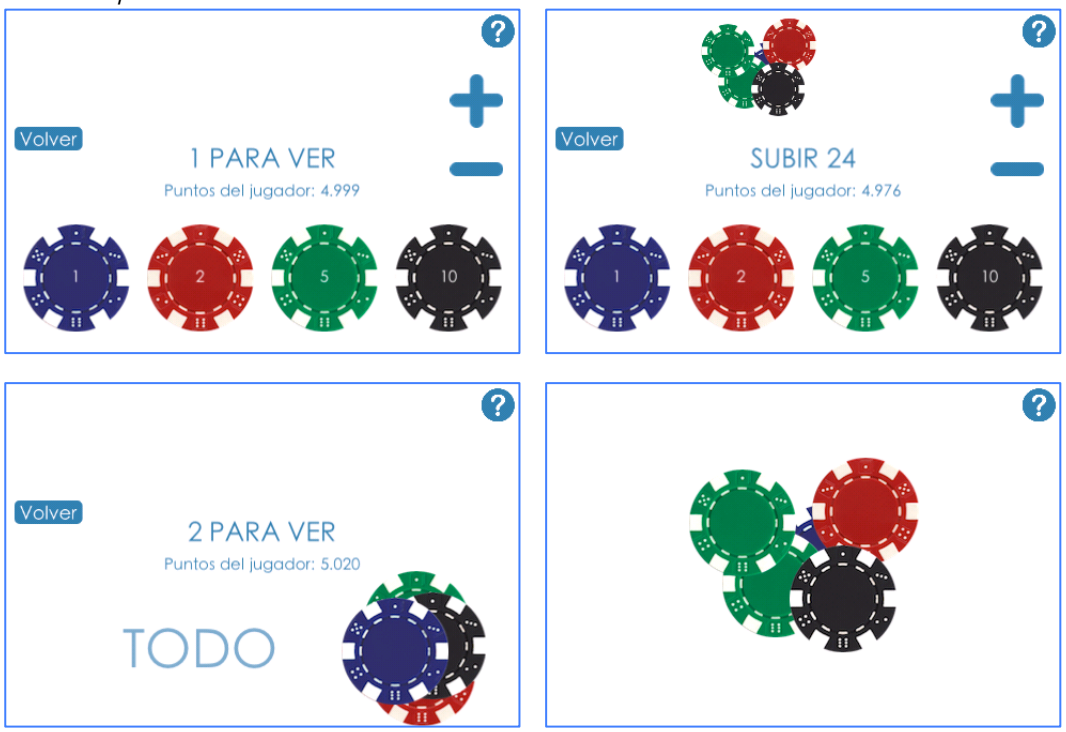

Imagen 6.15

La pantalla de apostar permite al usuario escoger la cantidad de puntos que quiere realizar la apuesta. La pantalla informa al usuario de los puntos que le quedan en cada momento y los puntos que necesita para igualar la apuesta. Si el usuario desplaza una ficha hacia arriba, aumentará los puntos de la apuesta que quiere realizar. Cada ficha de la pantalla tiene un valor y ese valor se puede modificar mediante los botones de + y - .Si el usuario desplaza la primera ficha de la izquierda hacia la ultima ficha de la derecha y después la desplaza hacia arriba, el usuario estará apostando todos sus puntos. Una vez que el usuario quiere confirma la apuesta, solamente tendrá que dar sobre las fichas que se han lanzado y, tras una animación que centrará y ampliará las fichas, él podrá desplazarlas hacia arriba para realizar la apuesta.

6.4.13. Finalización de la mano

Imagen 6.16

La vista de finalización de la mano informa al usuario que la mano actual se ha terminado con uno o varios ganadores. Si se retiran todos los jugadores menos uno, la pantalla solo informará que tal jugador ha ganado el bote. Si todas la rondas de apuestas se han igualado o si se han realizado all-ins, la pantalla mostrará las cartas comunitarias, las cartas de cada jugador con la identificación de su jugada y informará de quien es el ganador o ganadores de la mano. En caso que haya un ganador del juego, la pantalla lo mostrará. La vista es desplazable verticalmente mediante el gesto de desplazamiento. En la parte inferior de la pantalla habrá un botón que todos los jugadores del juego deben tocar para iniciar la siguiente mano.

6.5. Sonidos e idiomas

Otros aspectos importantes en la interfaz gráfica son el uso de diferentes sonidos, como por ejemplo una música de fondo o un sonido de una carta, y la traducción de la interfaz en varios idiomas.

Sonidos

Para realizar la tarea de reproducir los diferentes sonidos del juego se ha utilizado la librería CocosDenshion. Esta librería forma parte de la librería de Cocos2d y proporciona dos clases que son SimpleAudioEngine y CDAudioManager.

La clase SimpleAudioEngine proporciona los métodos necesarios para la reproducción de un solo sonido o música (que como mucho se puede ir repitiendo) y con pocas posibilidades de configuración del sonido. En cambio, la clase CDAudioManager proporciona los métodos que

permiten la gestión de varios sonidos o músicas y con más posibilidades de configuración que la clase SimpleAudioEngine.

Idiomas

La traducción de la interfaz se realiza mediante el uso del método NSLocalizedString de la librería Foundation. Esté método funciona mediante el mecanismo de clave – valor, que proporcionando una clave tipo string proporciona un valor tipo string. El valor proporcionado por este método no varia solo por la clave dada, sino también por el idioma actual del sistema operativo iOS. Por lo tanto, para una misma clave, el método puede dar diferentes valores según el idioma. Los valores proporcionados por este método están almacenados en un archivo de texto plano en la carpeta localizable.strings. Cada archivo de esta carpeta representa un idioma. En el caso que un idioma no tenga su respectivo archivo de traducción, el método devolverá por defecto el valor en ingles.

6.6. Pruebas

Las pruebas que se han realizado en la capa de presentación consisten primero en pruebas unitarias de cada una de las pantallas y después, en varias pruebas con la lógica general del juego.

En las pruebas unitarias que se han realizado en cada una de las pantallas, se ha comprobado que el funcionamiento de la interacción del usuario y la pantalla sean correctas, y que la pantalla muestre sus respectivos eventos de forma correcta.

En las pruebas con la lógica del juego se han comprobado que la gestión de las transiciones de una pantalla a otra sean las correctas cuando estas son provocadas por el usuario y por los demás jugadores del juego. También se han comprobado que desde cada una de las pantallas del juego se puedan cambiar a la vista de “Lista de reconexión” al recibir el evento de desconexión de un jugador.

7. PLANIFICACIÓN Y COSTES

Acabados los capítulos de desarrollo y cumplidos los diferentes objetivos de este proyecto final de carrera, a este documento solo le faltan los capítulos de evaluación del proyecto, que son los siguientes: planificación y costes, y conclusiones.

7.1. Introducción

En este capítulo se evalúan dos aspectos importantes del proyecto que tienen un enfoque diferente a todo lo que se ha visto en este documento. Los dos aspectos a evaluar son los que conciernen a la gestión empresarial y económica del proyecto.

La parte empresarial observa la **planificación**, la ejecución y la duración del proyecto.

En cambio, la parte económica observa los recursos necesarios para la ejecución del proyecto y sus diversos **costes** derivados por su uso.

7.2. Planificación

Al iniciar un proyecto, una de las primeras cosas que se deben hacer antes de ejecutarlo es una planificación. La planificación tiene varios objetivos que ayudan a la correcta ejecución del proyecto. Los objetivos de la planificación han sido los siguientes:

- Desglosar los diferentes objetivos / desarrollos en diversas tareas más pequeñas.
- Determinar la duración de cada tarea.
- Contextualizar la planificación en una línea temporal, encadenando (y solapando) las diversas tareas, y determinar el inicio y el fin del proyecto.

Una vez realizada la planificación, la ejecución del proyecto es más sencilla porque en cada momento se sabe que tarea se debe hacer y el tiempo que se dispone para hacerlo.

El problema de las planificaciones es que rara vez se cumplen fielmente a lo planificado porque es difícil prever todos los acontecimientos futuros que se pueden producir en un espacio de tiempo. Los principales factores que pueden alterar una planificación son diversos: humanos, económicos, etc.

Por esta razón, una vez terminado un proyecto es adecuado analizar cuanto tiempo en total se ha desviado y cuales han sido los factores que han alterado la planificación. Este post análisis es muy útil para futuros proyectos porque el conocimiento adquirido por el análisis ayudará a ajustar mejor las futuras planificaciones.

7.2.1. Planificación inicial

La realización de la planificación de este proyecto se basó en los objetivos del proyecto y en los objetivos personales establecidos en el capítulo 1. A partir de estos objetivos y del conocimiento que se iban a realizar dos desarrollos diferentes, se dividió toda la carga de trabajo en diferentes tareas y, en cada una de las tareas, se estimó una cantidad de días de trabajo. Se estableció que cada día equivaldría 4 horas de trabajo. Las tareas planificadas fueron las siguientes:

- Documentarse sobre la librería Game Kit. 3 días
- Investigar diversos aspectos del Game Kit. 4 días
- Desarrollo de la librería P2P
    - Definición del sistema y análisis de requisitos. 2 días
    - Especificación. 3 días
    - Diseño. 5 días
    - Implementación. 5 días
    - Pruebas. 7 días
- Desarrollo de la aplicación
    - Definición del sistema y análisis de requisitos. 3 días
    - Especificación. 4 días
    - Diseño. 5 días
    - Implementación. 10 días
    - Pruebas. 10 días
- Desarrollo de la interfaz gráfica – cocos2d
    - Diseño. 5 días
    - Implementación. 15 días
    - Pruebas. 10 días
- Documentación. 30 días

La “Interfaz gráfica” se dividió del grupo de tareas de “Desarrollo de la aplicación”, creando un nuevo grupo, porque la carga de trabajo era equivalente a los demás grupos de tareas.

También se estimó que habrían una serie de días en que las tareas se podrían solapar porque el desarrollador podría realizar cambios de dinámica para que el trabajo fuera más ameno, sobre todo cuando se llevarán varios días realizando la misma tarea.

La carga total de trabajo estimado entre todas las tareas es de 121 días, menos 9 días de solapamiento, dan un total de 112 días de trabajo.

Ambos diagramas de Gantt inician desde la misma fecha. Si el proyecto hubiera sido continuo, la finalización habría sido el 4 de septiembre del 2010. El número de días de trabajo real han sido de 167 días que transformados en horas son unas 668 horas de duración del proyecto

La imagen 7.1, que está en la siguiente página, traslada todas las tareas enumeradas anteriormente en una línea temporal en forma de gráfica de Gantt. En el diagrama se tomó como

fecha de inicio el 11 de enero del 2010 y estaba planificada su finalización para el 17 de junio del 2010.

7.2.2. Planificación final

La ejecución del proyecto ha sido muy discontinuado debido a razones personales. Pero aparte de estas razones, también hubo otros problemas que no se tuvieron en cuenta en el momento de realizar la planificación inicial:

- La dificultad de reunir como mínimo tres dispositivos con el sistema operativo iOS para realizar las pruebas reales.
- Las diferencias de comportamiento de los simuladores de iphone, conectados en una red local, respecto a los dispositivos reales conectados por Bluetooth.
- Errores en la estimación del tiempo de las tareas de implementar y de realizar las pruebas en el desarrollo de la aplicación y en el desarrollo de la interfaz de usuario. El desconocimiento del lenguaje de programación, del iOS SDK y de la librería Cocos2d son la principal razón del error de estimación.

A causa de la discontinuidad del proyecto es muy difícil dibujar un diagrama de Gantt con el seguimiento real del proyecto. Para solucionar este problema, en las imágenes 7.2, 7.3 y 7.4 se han dibujado un diagrama estimado sin los lapsos de tiempo (continuo). En las imágenes se indican en negro las tareas que han sido modificadas respecto a la planificación inicial.

Ambos diagramas de Gantt inician desde la misma fecha. Si el proyecto hubiera sido continuo, la finalización habría sido el 4 de septiembre del 2010. El número de días de trabajo real han sido de 167 días que transformados en horas son unas 668 horas de duración del proyecto.

El análisis del desvío entre la planificación inicial y la real es la siguiente:

Grupos de tareas	Planificado	Real	Diferencia
Desarrollo de la librería P2P	27	64	37
Desarrollo de la aplicación	29	38	9
Desarrollo de la interfaz de usuario	28	37	9
Documentación	28	28	0
TOTAL	112	167	55

Se puede observar que la tarea que más se ha desviado es el desarrollo de la librería P2P con 37 días de más.

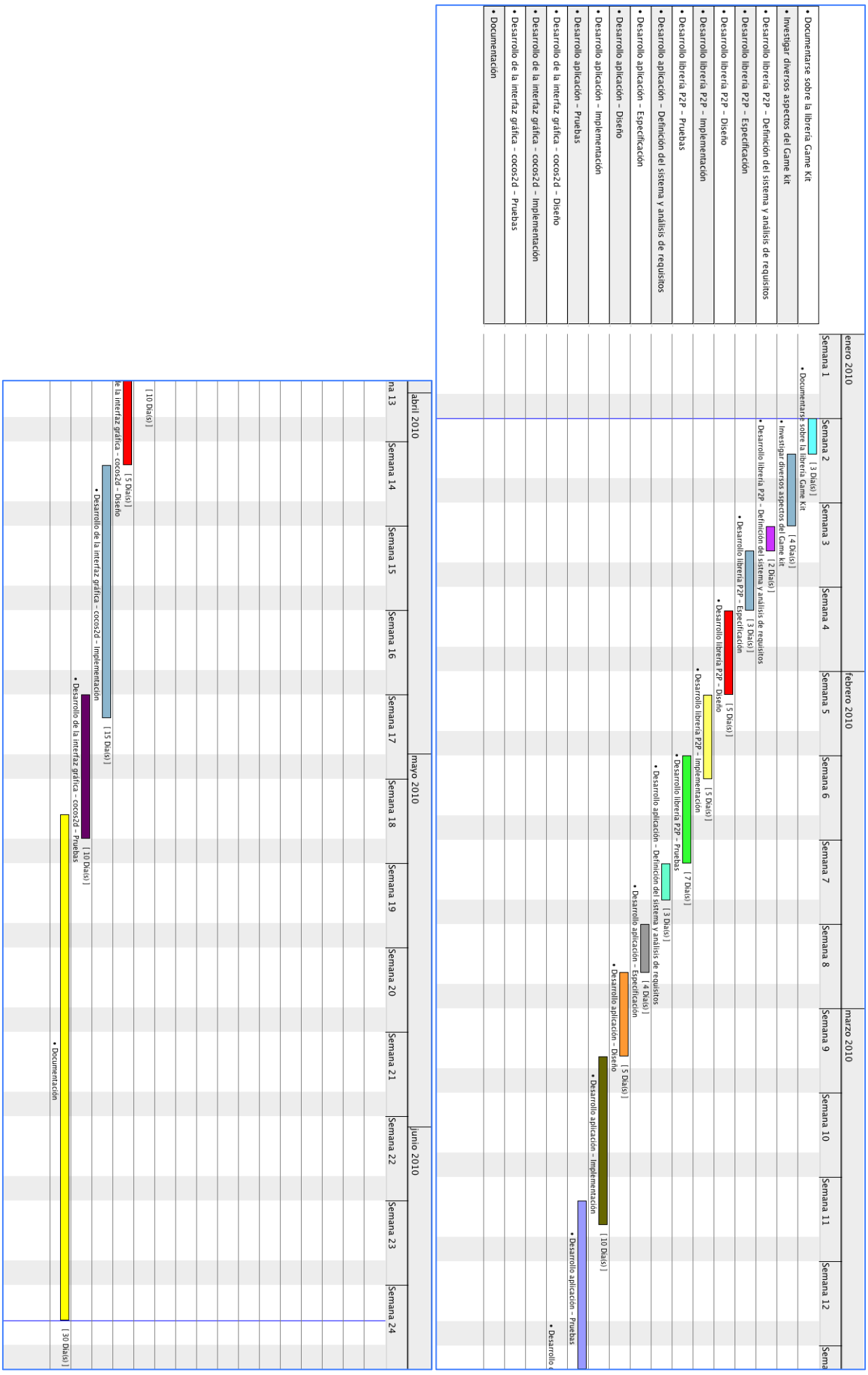

Imagen 7.1 – Planificación inicial

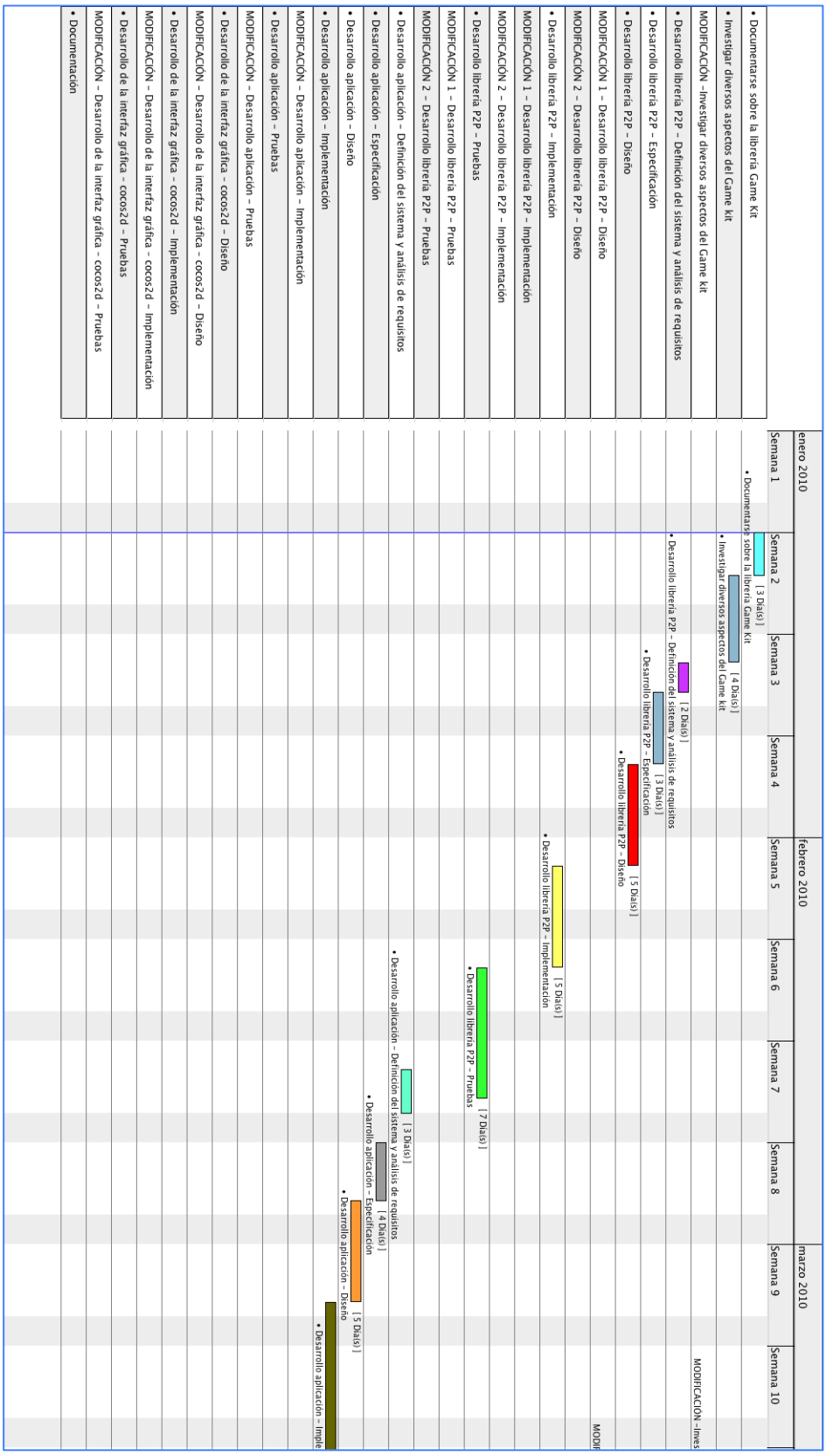

Imagen 7.2 – Planificación estima, parte 1

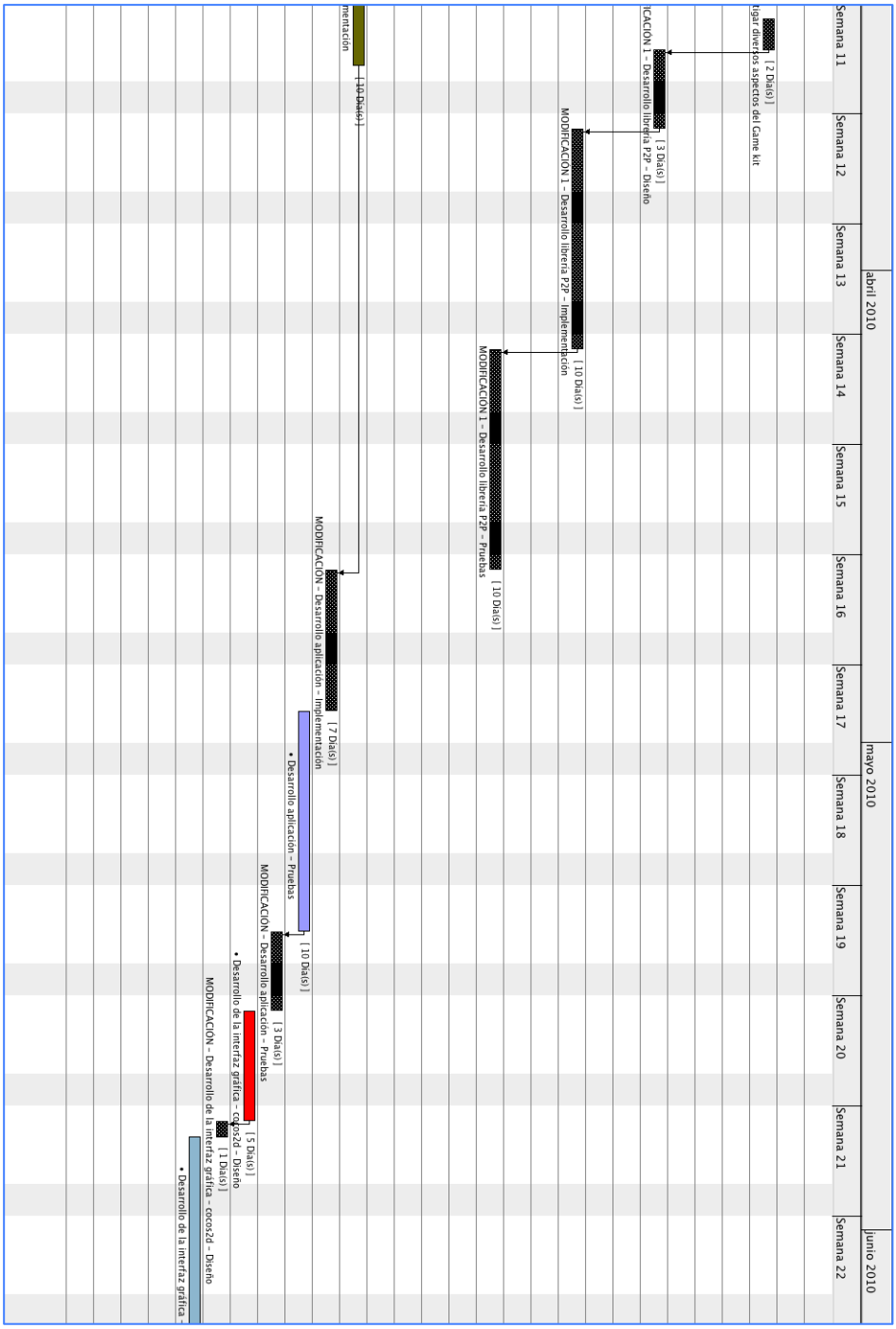

Imagen 7.3 – Planificación estima, parte 2

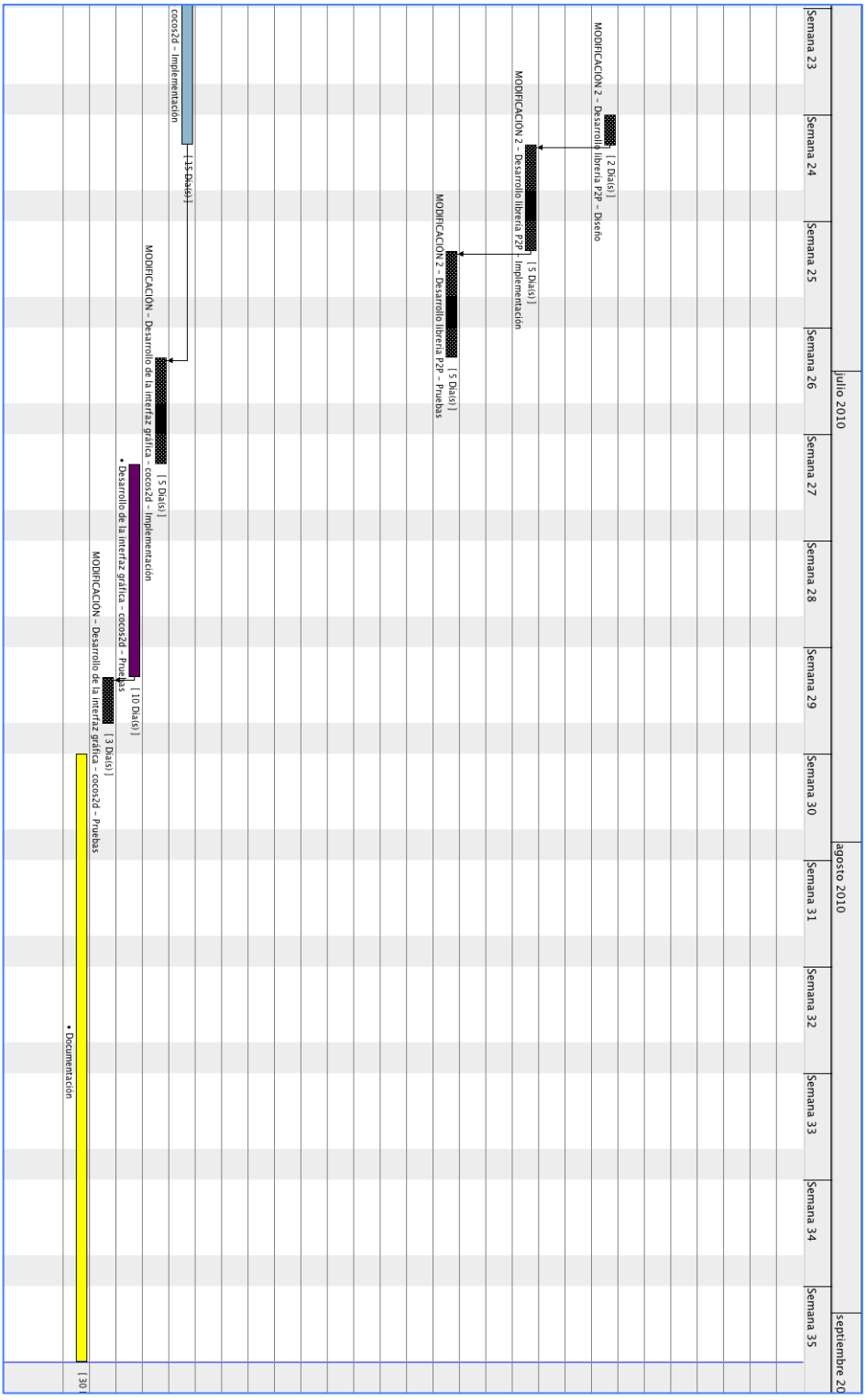

Imagen 7.4 – Planificación estima, parte 3

Como última observación, la duración real de este proyecto, de 668 horas, se aproxima bastante a la duración de un proyecto de la ingeniería superior.

7.3. Costes

El objetivo de este apartado es poner un precio a la ejecución del proyecto en el caso que éste fuese realizado por una empresa. Para realizar esta tarea es necesario asignar los recursos necesarios y, a partir de ellos, estimar los diversos costes.

7.3.1. Recursos

Recursos humanos

El primer recurso que siempre se estima es el humano porque es el que genera mayor coste. Normalmente, un equipo de trabajo está formado por los siguientes miembros: un **jefe de proyecto**, un **analista** y un **programador**. A parte, es normal incorporar temporalmente otras personas de otras especialidades. En este caso, es necesario la incorporación de un **diseñador gráfico** para crear las imágenes necesarias para la interfaz gráfica.

Recursos hardware

Los recursos hardware necesarios para el equipo de trabajo y por el diseñador gráfico son los siguientes:

- 4 iMac.
- 1 tabla digitalizadora.
- 3 dispositivos con sistema operativo iOS diferentes: iPhone, iPod Touch y iPad.

Recursos software

Los recursos software para el hardware de la lista anterior son los siguientes:

- 4 licencias de Mac OS X.
- 4 licencias del paquete ofimático Microsoft Office.
- 1 licencia Adobe Photoshop.
- iOS SDK.
- Librería Cocos2d.

Otros recursos

Existen varios recursos que no entran directamente en las otras categorías, estos recursos son los siguientes:

- Línea ADSL.
- Licencia de desarrollador iOS.

7.3.2. Costes

A partir de los recursos descritos anteriormente, se estima el coste de cada uno en base de su utilización para en la ejecución del proyecto. Hay que tener presente que no todos los recursos tienen coste.

Coste humano

Como punto de partida hay que estimar los salarios de cada uno de los miembros del equipo y del diseñador, calcular su coste para la empresa teniendo en cuenta que hay que pagar la seguridad social de cada uno y obtener su coste por horas. Los cálculos se basan en que un trabajador trabaja una media de 240 días al año (8 horas/día) y que el coste de la seguridad social de un trabajador es más o menos el 32 % de su salario bruto.

Trabajador	S. bruto anual	Sueldo + S.S.	Coste/día	Coste/hora
Jefe de proyecto	40.000 €	52.800 €	220 €	27,5 €
Analista	27.000 €	35.640 €	148,5 €	18,56 €
Programador	20.000 €	26.400 €	110 €	13,75 €
Diseñador gráfico	18.000 €	23.760 €	99 €	12,37 €

Por otra parte, se organiza el trabajo del proyecto en diferentes familias para después distribuirlo por los trabajadores. Los datos se basan en la planificación inicial de 448 horas:

Tareas	Días	Horas
Definición del sistema y a. de requisitos	12	48
Especificación	7	28
Diseño	15	60
Implementación	26,5	106
Pruebas	23,5	94
Documentación	28	112
TOTAL	112	448

El reparto de las tareas es la siguiente:

Tareas / Trabajadores	Jefe p.	Analista	Programador	Diseñador g.
Def. del sistema y a. de req.	24 horas	24 horas	0 horas	0 horas
Especificación	14 horas	14 horas	0 horas	0 horas
Diseño	0 horas	60 horas	0 horas	0 horas
Implementación	0 horas	0 horas	95,4 horas	10,6 horas
Pruebas	0 horas	9,4 horas	84,6 horas	0 horas
Documentación	78,4 horas	11,2 horas	22,4 horas	0 horas
TOTAL	116,4 horas	118,6 horas	202,4 horas	10,6 horas

Ahora que se ha obtenido la cantidad de horas de trabajo de cada uno, se calculará el coste humano de cada trabajador:

Trabajador	Horas	Coste/hora	Coste	% Coste
Jefe del proyecto	116,4 horas	27,5 € / hora	3.201 €	38,5 %
Analista	118,6 horas	18,56 € / hora	2.201,2 €	26,47 %
Programador	202,4 horas	13,75 € / hora	2.783 €	33,46 %
Diseñador gráfico	10,6 horas	12,37 € / hora	131,1 €	1,57 %
TOTAL	8.316,3 €			

Coste del hardware

El coste del hardware se basa en el cálculo de la amortización del equipo por el tiempo usado en el proyecto. La amortización de un ordenador y una tabla digitalizadora son de 3 años y la de un dispositivo iOS es de 2 años. El uso de los equipos han sido 4 meses.

Recurso	Coste	Plazo amorti.	% imputable	Coste impu.
4 iMac	4*1145 € = 4580 €	36 meses	11,1 %	508,38 €
1 tabla d. Wacom Intuos5	224,9 €	36 meses	11,1 %	24,96 €
1 iPhone 4S 16 GB	599 €	24 meses	16,6 %	99,43 €
1 iPod Touch 8 GB	185 €	24 meses	16,6 %	30,71 €
1 iPad 3 WiFi 16 GB	479 €	24 meses	16,6 %	79,51 €
TOTAL	742,99 €			

Coste del software

El cálculo del coste del software se procede de la misma manera que el del hardware. La amortización es de 3 años.

Recurso	Coste	Plazo amor.	% impu.	Coste impu.
4 licencias Mac OS X	4*30 € = 120 €	36 meses	11,1 %	13,32 €
4 licencias M. Office	4*139,95 € = 559,8€	36 meses	11,1 %	62,14 €
1 licencia de A. Photoshop	189,95 €	36 meses	11,1 %	21,08 €
1 iOS SDK	0 €	36 meses	11,1 %	0 €
1 librería Cocos2d	0 €	36 meses	11,1 %	0 €
TOTAL	96,54 €			

Otros costes

Recurso	Coste	Plazo amor.	% impu.	Coste impu.
1 línea ADSL	29,95 € * 4 meses = 119,8 €	4 meses	100 %	119,8 €
1 licencia de desarrollador iOS	99 $ = 78,3 €	12 meses	33,3 %	26,07 €
TOTAL	145,87 €			

7.3.3. Coste final

El desglose de los costes finales del proyecto es la siguiente:

Concepto	Coste
Coste humano	8.316,3 €
Coste del hardware	742,99 €
Coste del software	96,54 €
Otros costes	145,87 €
TOTAL	9.301,7 €

El coste final del proyecto es de **9.301,7 €**.

8. CONCLUSIONES

En este último capítulo del documento el objetivo es evaluar el proceso y la finalización del proyecto, desde el punto de vista académico y desde el punto de vista personal. Los dos puntos de vista engloban todo el conocimiento adquirido durante la ejecución del proyecto final de carrera.

A parte, se ha añadido un tercer apartado con el fin de explicar las posibles mejoras y ampliaciones del proyecto.

8.1. Conclusión del proyecto

Una vez concluidos los desarrollos de la librería y de la aplicación, se puede analizar el trabajo realizado y comprobar si los desarrollos cumplen los objetivos establecidos en el primer capítulo.

La librería P2P desarrollada cumple todas las características que se pedían en el primer objetivo. La librería trabaja sobre la librería Game Kit, permitiéndole realizar conexiones punto-a-punto (peer-to-peer) entre dispositivos. Las redes creadas por la librería pueden contener más dispositivos que las creadas por la librería Game Kit. Cuando hay una desconexión en la red, ésta puede recuperarse y continuar la comunicación. Y por último, la librería está encapsulada, convirtiéndose en un componente reusable en diferentes aplicaciones.

En la segunda parte del proyecto se ha desarrollado una aplicación desde cero la cual debía tener la necesidad de crear una red de dispositivos. La aplicación desarrollada ha sido un juego de póker que utiliza la librería P2P para poder comunicarse con los demás jugadores del juego. Se ha podido verificar el correcto funcionamiento del juego y, por lo tanto, el correcto funcionamiento de la librería. Comprobado la viabilidad de librería P2P, se ha cumplido el segundo y último objetivo del proyecto.

8.2. Conclusión personal

Este proyecto me ha servido para aprender a desarrollar por primera vez en una plataforma móvil. Llevaba tiempo queriendo aprender sobre ello y la realización de este proyecto me ha permitido conseguirlo.

Al desarrollar en la plataforma de Apple, no solo he tenido que aprender parte del iOS SDK sino que también me ha permitido aprender un nuevo lenguaje de programación (Objective-c). A parte de los frameworks del SDK oficial, también he aprendido a utilizar un framework open source de generación de gráficos.

Además de cumplir los objetivos personales que me había propuesto antes de iniciar el proyecto, he asimilado aún más la importancia de la capacidad de aprender cosas nuevas por

cuenta propia y ponerlas en práctica inmediatamente en un desarrollo concreto. Y, por lo tanto, pienso que la experiencia vivida al realizar este proyecto me será muy útil a la hora de realizar futuros trabajos.

8.3. Mejoras y ampliaciones

Durante las diversas pruebas que he podido realizar en la librería P2P, he podido observar que el sistema de reconexión automática de los dispositivos aún conectados no siempre funciona. Creo que éste aspecto podría ser mejorado realizando la búsqueda de nuevas estrategias en el diseño de la red y experimentado aun más con el framework Game Kit.

A parte en la aplicación del juego de Texas Hold'em hay un sin fin de posibles ampliaciones que habrían sido muy interesantes tenerlas implementadas, como por ejemplo, la capacidad de guardar partidas, la inclusión de jugadores no humanos o la integración del juego al Game Center para aumentar el aspecto social.

9. AGRADECIMIENTOS

Quiero agradecer a todos el apoyo que me han ofrecido durante todo este tiempo, en especial:

A mi familia y mi novia que han estado día a día ayudando me.

A mi hermana por ser la primera lectora.

A mi amigos por recordarme que hay que terminar lo que se empieza.

Y a todos los que me han dejado dispositivo iOS para hacer las pruebas.

10. BIBLIOGRAFÍA

Libros

- Dave Mark and Jeff LaMarche . Beginning 3 iPhone Development – Exploring iPhone SDK. 2009
- Steffen Itterheim . Learn iPhone and iPad cocos2d Game Development . 2010
- Fernando López Hernández . El lenguaje Objective-C para programadores C++ y Java . 2008
- Fernando López Hernández . Programación Cocoa con Foundation Framework . 2009
- Dolors Costal – Xavier Franch – M. Ribera Sancho – Ernest Teniente . Enginyeria del software – Especificació – Especificació de sistemes orientats a objectes amb la notació UML , 2005

Webs

- MacProgramadores - http://macprogramadores.org/
- iOS Dev Center – Apple - http://developer.apple.com
- Apple España – http://apple.com/es/
- Game Kit Programming Guide - http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/GameKit_Guide/GameKitConcepts/GameKitConcepts.html#//apple_ref/doc/uid/TP40008304-CH100-SW1
- Bluetooth Core Version 2.0 + EDR - https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=40560
- Cocos2d for iPhone - http://www.cocos2d-iphone.org/
- Stackoverflow - http://stackoverflow.com/
- Pokerstarts España - http://www.pokerstars.es/poker/games/texas-holdem/ http://www.pokerstars.es/poker/games/rules/